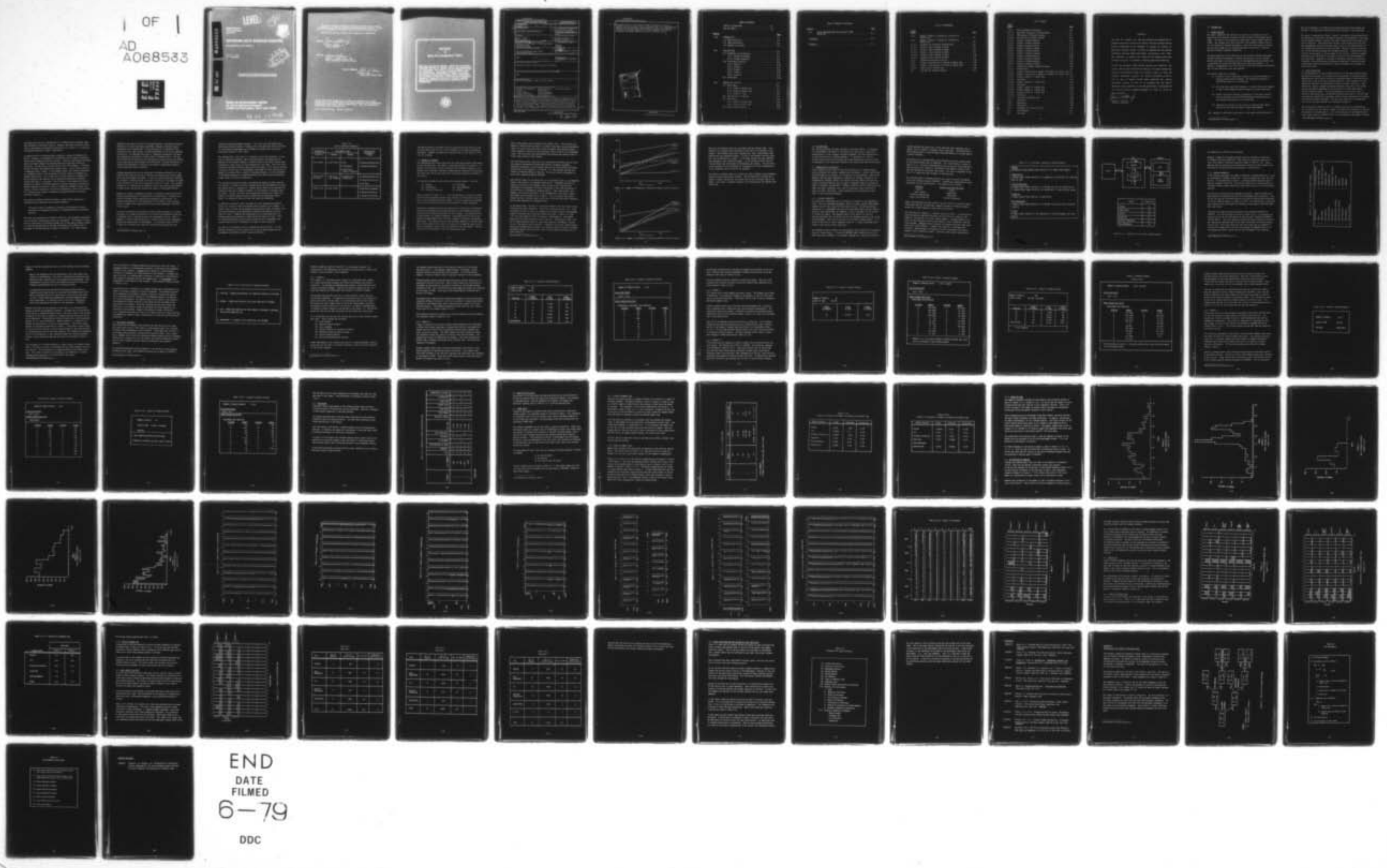


RADC-TR-79-67
Final Technical Report
March 1979

SOFTWARE DATA BASELINE ANALYSIS

General Electric/C & I Systems

Dr. D. L. Fish
M. T. Matsumoto

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

APPROVED: James V. Cellini, Jr.
JAMES V. CELLINI, Jr.
Project Engineer

APPROVED: Wendall C. Bauman
WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: [illegible]
JOHN P. HUSS
Acting Chief, Plans Office

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 **REPORT DOCUMENTATION PAGE** | READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 18 RADC-TR-79-67	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 SOFTWARE DATA BASELINE ANALYSIS		5. TYPE OF REPORT & PERIOD COVERED 9 Final Technical Report 14 Nov 77 - 17 Nov 78
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) 10 Dr. D. L. Fish M. T. Matsumoto		8. CONTRACT OR GRANT NUMBER(s) 15 F30602-78-C-0022
9. PERFORMING ORGANIZATION NAME AND ADDRESS S/C General Electric/Command & Information Systems 450 Persian Drive Sunnyvale CA 94086		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55812005
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441		12. REPORT DATE 11 Marc[redacted]979
		13. NUMBER OF PAGES 74 12 84p.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same 16 5581 17 20		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: James V. Cellini (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software error baselines	linear regression	structural analysis
software	AID analysis	
software errors	software analysis	
software reliability	error distributions	
software error typology	data collection	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The subject report summarizes the results of an analysis of software error data supplied by the Information Sciences Division of Rome Air Development Center (RADC). These data consisted of the software problem histories of five large-scale software developments, individually collected by the development contractors and supplied to RADC. The problems were classified by a previously developed error typology. The purpose of the analysis was to investigate the existence of any consistencies in the occurrence of errors utilizing the five

CONT

DD FORM 1 JAN 73 1473

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

9R 390151

JOB

Con't

development efforts. The analysis included consideration of the error typology, rate of occurrence, time of occurrence, time to fix, and module size. Results of the analysis isolate methodological problems in the gathering of software error data and suggest that positive incentives be provided to development team members involved in the data collection effort.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

Evaluation

The need for reducing the cost and increasing the productivity of software within the Air Force still exists. This task involved various areas to investigate. It was necessary to surface the results of particular previous efforts in order to coordinate all the valuable information from them toward these needs. Particularly, this effort was undertaken to analyze the results of five software error data collection projects in an attempt to develop quantitative baselines.

It fits into the goals of RADC TPO-R5A, Software Cost Reduction; Sub Thrust Software Data Collection and Analysis. The report presents the results of the analysis of data from different types of large DOD software development projects. The value of this effort is that it will be used to support current model prediction and quality measurement projects as well as be evaluated with the goal of developing useful baselines. It has been significant in bringing forth the areas that require in-depth development in order to arrive at these baselines.

James V. Cellini Jr

JAMES V. CELLINI, Jr
Project Engineer

1.0 INTRODUCTION

1.1 REPORT OVERVIEW

This report summarizes the results of an analysis of software error data supplied by the Information Sciences Division of Rome Air Development Center (RADC). The analysis was performed for RADC under contract number F30602-78-C-0022. The software error data consisted of the software problem histories of five large-scale software developments, which were individually collected and provided to RADC by five software development contractors, ([THAT76], [WILH77], [FRIM77], [BAKW77], [RYEP77])*.

The major objectives of the study were to utilize these software development problem histories to determine if certain characteristics of the software exhibit consistent ralationships with the corresponding problem histories and to determine the validity and applicability of these relationships. In addition, recommendations for future analysis which would further the establishment of these baselines, were also expected.

The approach taken was as follows:

(1) Establish a set of functional categories into which elements of a software system could be grouped. The categories presented in [THAT76]* were used as a starting point.

(2) Utilizing these functional categories , classify the various modules of each of the software systems provided in the error data base.

(3) Perform statistical analyses to determine if consistent relationships or baselines can be established between characteristics of the software and measures of its reliability.

(4) Determine the validity of the results by assessing their applicability to error data from other software developments.

This approach is described in more detail in the report and definitions for

*See References following page 4-3.

many of the measures of reliability and characteristics of the software are provided. The constraints imposed by the data available is also described.

The report is organized as follows: Section 1 provides an overview describing the objectives of the study, the general approach, the perspective of the study, and the general findings. Section 2 provides a description of the data available for analysis. This description includes a brief discussion of the software developments from which the data came, as well as the characteristics of the software and types of error data provided. Section 3 contains a description of the analyses performed and the detailed results. A discussion of the validity of the results is also provided in this section. Section 4 suggests what data can be collected in the future to assist in the establishment of error baselines. Appendix A-1 provides a structural analysis using the AID (Automatic Interaction Detector) technique to determine if the method could provide some insight into the effect of certain parameters on the number of errors which occur.

1.2 STUDY PERSPECTIVE

In the acquisition of a new software system, one of the major problems facing the acquisition manager is the prediction and assessment of software quality. Among the many factors which contribute to the measurement of software quality, reliability is one of the most important [MCCJ77]*. Until recently, no techniques were available to quantitatively measure software reliability. Reliability was largely a subjective measure provided by the users of the system and was not readily comparable to the reliability of other functionally similar software systems. A direct consequence of this void was often to delay the realization that a reliability problem existed until it was too late to achieve any substantial improvement except for the correction of the obvious, high-priority software problems. The users were often left to contend with less serious software problems with workaround procedures.

The occurrence of software errors is a primary indication of unreliability; but, reliability is only one of a number of factors which contribute to the overall measure of software quality. To a certain degree, the contribution which these factors make to software quality is also measured by the number of errors,

*See References following page 4-3.

indicating that the error characteristics of software are an extremely important indicator of the overall quality. However, the effect of errors is most clearly indicated by the reliability factor and, as a result, it is receiving considerable attention in RADC's study efforts.

A concerted effort is now being made to develop a concept of software reliability. However, a reorientation in perspective must be made by individuals familiar with hardware reliability concepts. There are significant differences which distinguish hardware and software when visualizing the reliability discipline. At the outset, software does not fail like hardware. A hardware failure indicates that something has changed states - from a working state to a nonworking state. Rather, in the software domain, the condition equivalent to a hardware failure is the occurrence of an error which is analogous to a hardware design error. For a given set of initial conditions, software will always accomplish the identical set of operations producing the identical results each time it is executed. This of course is not necessarily true of hardware and imparts a different meaning to the basic measure of reliability. Another important distinction is that in correcting a hardware failure, the system is normally restored to its initial configuration, while the correction of a software error produces a different configuration which will exhibit different properties.

The concept of software reliability which is used in this study which is supported by past experience can be simply stated as:

> The extent to which a program or collection of functionally related programs can be expected to perform its intended function with required precision.

With this basic definition of software reliability, two fundamental approaches to its practical interpretation need to be considered. The simplest of these, which is applicable to individual programs and to some extent software systems, is a measure characterized by the Mean Time Between Error (MTBE) which is analogous to the MTBF measure of hardware reliability. For large software systems as are typical of many Air Force applications, the concept of Mission Reliability has more significance than MTBE, although a definite interrelationship exists between them. Mission Reliability is a measure of the probability that, once started, a stated operational mission can be completed successfully. An important point here is "completed successfully" and does not preclude the occurrence of certain types of software errors. This two level definition of reliability has been taken by others [LLOD77]* but differs by not requiring reliable software to be fault-free. Mission reliability is normally more meaningful in tactical and strategic systems where specific mission objectives must be achieved.

Although detected errors are an indication of software unreliability, a program with many known errors can be reliable and conversely one with no known errors can be extremely unreliable. It must be realized that the reliability of a program or software system is not only a function of the number of latent errors existing in it but also of the way in which it is used. Thus software reliability is a function of the number of errors, the severity and location of those errors, and the way in which the system is being used [MYEG76]*.

Attempts to develop a comprehensive theory of software reliability which will allow accurate prediction of software error characteristics, software availability, and other similar measures are beginning to show results. An essential contribution to the furtherance of this theory is the continued study of software error characteristics such as that described in Section 3 of this report.

The idea of achieving an environment in which reliable software is a normal occurrence is no longer unrealistic but reliability considerations must play an important part in the mainstream of the development activity. What is needed is a reliable method with which a software system can be evaluated at appropriate stages during its development. In a previous study [THAT76]* a large set of software error data was collected and analyzed from four

*See References following page 4-3.

separate software development projects. The initial work performed during that study and other sources of software error data have been used as the basis for the continued development and refinement of software error prediction techniques contained in this report.

The ultimate goal in this area is to develop a set of error baselines, in the form of regression equations, which accurately predict the expected error behavior of the software segments or modules within a functional category when estimated or actual values for the characteristics are input. With error baselines that have been validated against historical data, it will then be possible to predict, at the start of a development, the number of errors which would be typical of a module within a specific functional category. As the development of the module progresses, estimated characteristics could then be replaced by actual values to refine the prediction.

This information would be valuable in planning the amount of effort required for testing. It would also allow assessment during the development of how well the testing effort is progressing. Problem report trends can be compared with predicted values and a change in emphasis or reallocation of resources might be enacted. Finally, the error rate expected past delivery will impact the amount of resources planned during the operations and maintenance phase. Table 1.2-1 summarizes the use of the error baseline information.

In addition to the error rates, the types of errors expected and expected time to fix statistics that were derived from our analysis are valuable. If certain types of errors can be expected from particular types of modules, test plans and strategies can be generated to emphasize the detection of those types of errors. Standards and conventions can be established which are oriented toward the prevention of these particular types of errors. Plans for software operations and maintenance personnel skill requirements and training could also be influenced by the types of errors expected.

The time to fix estimates assist in planning the testing effort. It also provides indications of the response time to errors during operations and maintenance and therefore overall system availability.

Table 1.2-1
Uses of Baseline Information

INFORMATION FROM BASELINE ANALYSIS	DEVELOPMENT PHASE		OPERATION AND MAINTENANCE PHASE
	PLANNING	CONTROL	
Error Rates	● Test Effort	● Test ● Throughness ● Identified Areas of Emphasis	● Expected Reliability ● Required Resources
Distribution of Errors	● Test Plans and Strategy	● Standards and Conventions	● Expected Reliability ● Personnel Skill Mix ● Training
Time to Fix	● Test Effort		● Operations Response ● Required Resources ● System Availability

The data and results available from this study and previous efforts do not yet allow these types of uses of the information to be made with complete confidence. However, consideration should be given to this type of information for planning.

1.3 SUMMARY OF FINDINGS

The conclusions that can be drawn from this study were severely constrained by the available data. The impact is discussed in detail in section 2. In general, the inability to look at the data from different viewpoints, for example from a different functional categorization, prevented investigations that might have led to more significant correlations and more confidence in the results.

Six functional categories were defined for software modules. They are:

(1) Control
(2) Input/Output
(3) Pre/Post Processing
(4) Algorithm
(5) Data Management
(6) System

They are defined in Table 2.1.1-1. This categorization is similar to others which have been developed and have been used for classification of the modules in large command and control systems. As far as possible, the modules for each of the projects were classified according to these categories.

The analysis conducted was aimed at determining if statistical relationships could be found between certain characteristics of the software and characteristics of the problems reported with that software. The characteristics of the software, or software parameters, investigated and utilized in the analysis included module size, function, language, difficulty, and development method. The characteristics of the problems reported, or problem parameters, investigated and utilized in the analysis included the type of error, time of occurrence, severity and time required to fix the problem. Also an analysis of the confidence in the relationships was made.

Most of the analyses were conducted at the module level. This reflects the desire to identify characteristics at a module level which could be determined early in a project and could then be used to predict the problem characteristics expected. However the most consistent result found was at an aggregate level. This result was that approximately two problems per hundred lines of source code occurred in each project.

The consistency of this result was very interesting considering the fact that the projects represented different applications, different customers, different contractors, and the problem reports were from relatively different time periods in the projects' life cycles, i.e., the software had been submitted to different amounts of testing. This result closely corresponds to error rates reported elsewhere [NELR78]*.

One possible reason for the consistency at the aggregate level, is a phenomenon found in the analysis of programmer productivity. Programmer productivity figures are derived at an aggregate level because of the observed wide differences in programmers' abilities and because of the wide differences in the difficulty of implementing software modules. These same factors, programmers' ability and difficulty of the implementation, also have a significant impact on the reliability of a module. Thus at a module level these factors may have a greater impact on the error rate than functional categories and are only observed at an aggregate level.

At the module level, the analysis revealed differences in error rates for the different functional categories. These error rates are the baselines. Thus the number of lines of source in a module can be used to predict the expected number of problems that a particular module will have. Figures 1.2-1 and 1.2-2 give the regression lines for Project 1 and Project 3. The modules have been classified according to their function. Statistically, only a subset of these baselines exhibit a significant degree of confidence. The details of the analysis are in Section 3. However at this level, general observations can be made about baselines. For example, based on Project 3 the data management category baseline (error rate) is approximately twice that of all other categories.

*See References following page 4-3.

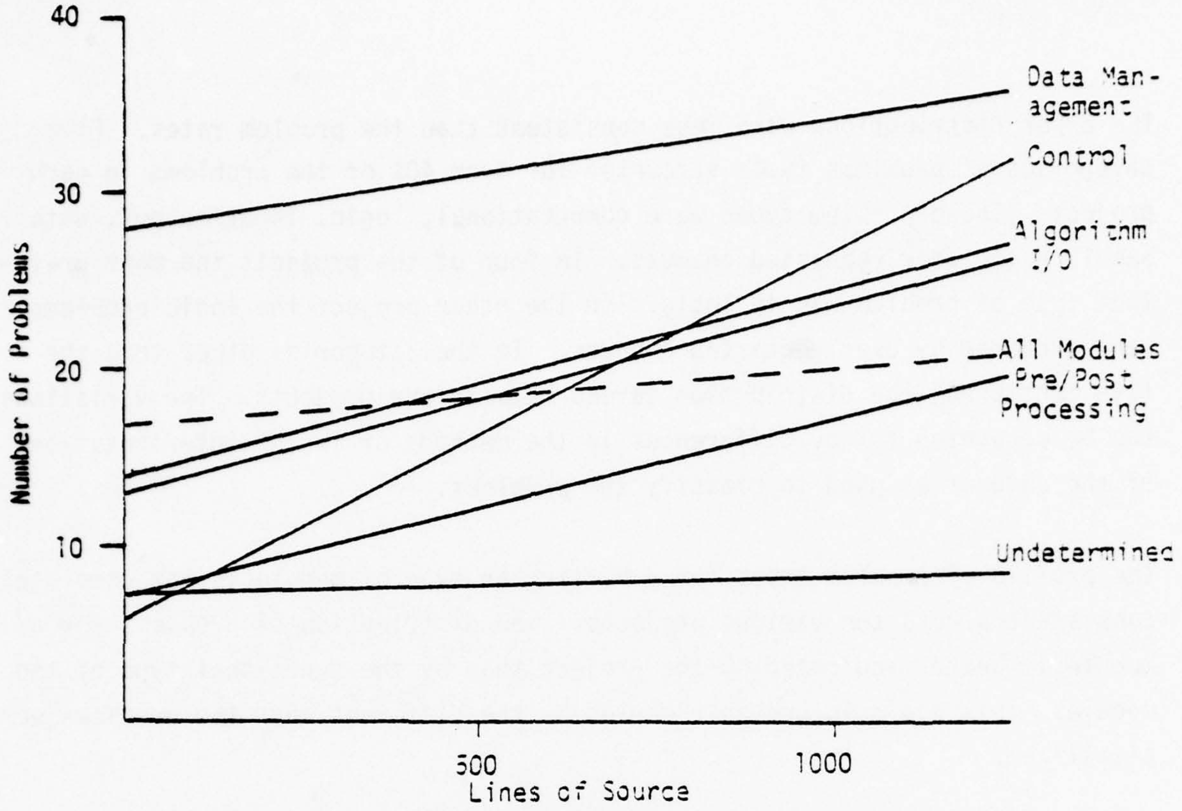


Figure 1.2-1 Number of Problems as a Function of Module Size for Project 3

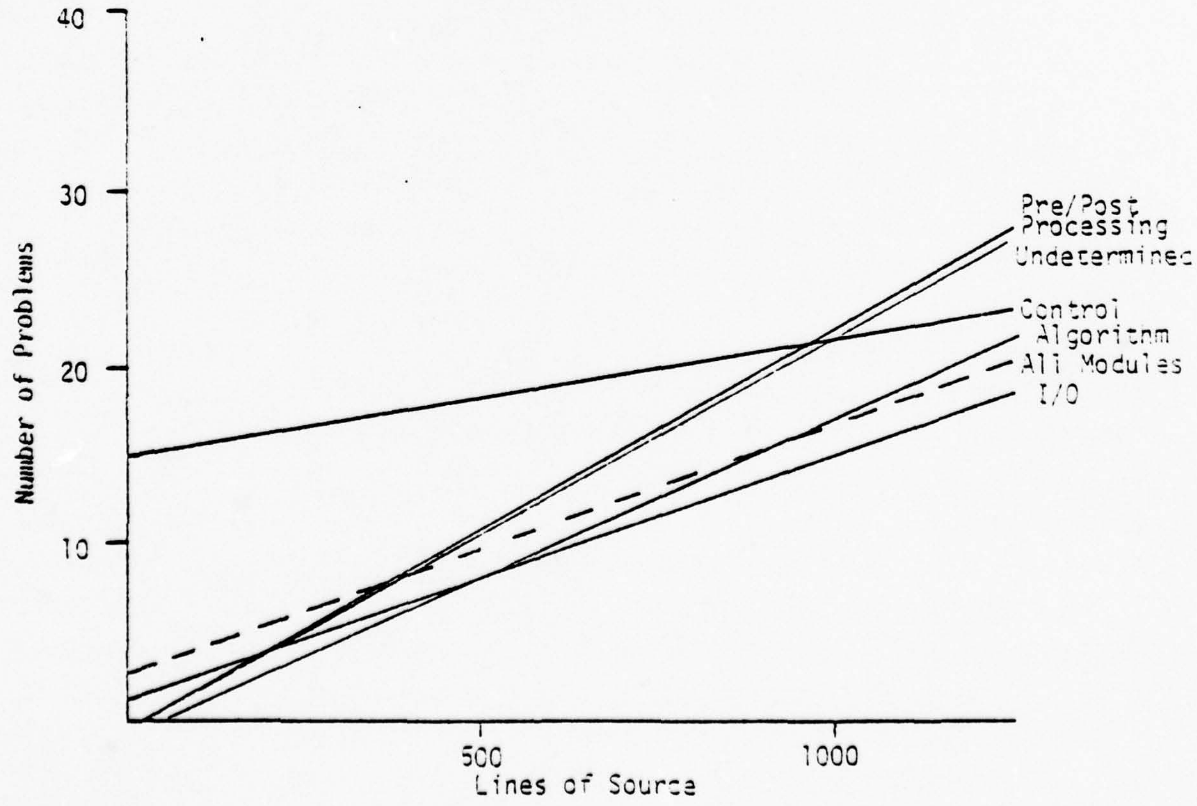


Figure 1.2-2 Number of Problems as a Function of Module Size for Project 1

The error distributions were less consistent than the problem rates. Five categories of problems types accounted for over 40% of the problems in each project. These problem types were computational, logic, input/output, data handling and user requested changes. In four of the projects the most prevalent type of problem was in logic. In the other project the logic problems were exceeded by user requested changes. In the categories other than the five mentioned, the distribution varied between the projects. The variations can be accounted for by differences in the methods or in the interpretation of the categories used to classify the problems.

The profile of problem types for a particular type of module is not completely consistent across the various projects. The distribution of problems for a module is better indicated by the project than by the functional type of the module. This again is probably caused by the different ways the problems were classified.

2.0 THE DATA BASE

This section describes the data available to the study effort. In paragraph 2.1 the parameters considered in the analysis are described including the characteristics of the software, software parameters, and the characteristics of the problem reports, problem parameters. In paragraph 2.2, the five software projects and the associated data about those projects are described In paragraph 2.3 the limitations imposed on the study by the data are discussed.

2.1 PARAMETERS OF THE ANALYSIS

A basic premise of this study is that the reliability of a software module can be predicted from intrinsic properties of the module. Thus by identifying certain properties of software, its reliability can be predicted. Some of the measures that have been suggested as predictors of software reliability are implementation language, module size, module function, module difficulty and certain structural measures such as number of branches, depth of nesting and number of operators and operands. Software reliability can be indicated by the number and type of problems, their time of occurrence, their difficulty to repair and their criticality. These parameters will be discussed in more detail in the following paragraphs.

2.1.1 SOFTWARE PARAMETERS

The selection of the proper unit of software for analysis is not immediately clear. An entire software development, which in a major project might exceed 100,000 lines of code, seems to be too coarse a unit. In practice certain subsections of a development are more error prone than others and the identification of these subsections, or segments, is one of the goals of the research in reliability theory. The approach taken in this study is to use the smallest meaningful unit of source text for the language processor used during the development. This unit of source will be called a software module. It is useful to use this as a basic component since an individual programmer would normally code and test these subsections.

The language in which a module is coded presents little difficulty in interpretation or identification. It might be FORTRAN, COBOL, JOVIAL or one of the other high level languages or an assembly language for a particular processor.

A minor problem that does occur is that some high level languages such as JOVIAL allow an intermix of assembly level instruction. The method used in the following report to specify these intermixed modules is to place them in a special category.

The function of a software module can be described by using a modification of the classification given in [WOLR74]*. This classification is shown in Table 2.1.1-1. The basic reasoning behind this particular classification is that the function of a module is determined by the module's effect on program and information flow within the system. This idea is expressed in Figure 2.1.1-1 where each type of module is characterized.

This classification is different than the one used to classify the modules in the five software projects [THAT76]*. A mapping was established to allow translation to this classification. This mapping was as follows:

[THAT76]*	Software Data Baseline Study
Control	Control
Input, Output	Input/Output
Primarily Computational	Algorithmic
Setup, Post Processing	Pre/Post-Processing

Other classifications may have proven to be more useful or provided a better statistical base for the baselines, however no means for reclassification except for a direct mapping as shown above was possible.

The difficulty of a module is a somewhat subjective matter. A categorization given by Wolverton [WOLR74]* describes the difficulty of a module as the number of interactions it has with system elements. An easy program is one with very few interactions with system elements, these include most applications programs. Medium difficult programs are programs that have some interaction with system elements. Examples are compilers, I/O packages and utilities. Hard programs are programs with many interactions with system elements such as operating systems. Certainly other factors contribute to

*See References following page 4-3.

Table 2.1.1-1 Functional Typology of Software Modules

CONTROL

AN EXECUTIVE MODULE WHOSE PRIME FUNCTION IS TO INVOKE OTHER MODULES

INPUT/OUTPUT

A MODULE WHOSE PRIME FUNCTION IS TO COMMUNICATE DATA BETWEEN THE COMPUTER AND THE USER

PRE/POSTPROCESSOR

A MODULE WHOSE PRIME FUNCTION IS TO PREPARE DATA FOR THE INVOCATION OF A COMPUTATIONAL MODULE OR AFTER THE INVOCATION OF A COMPUTATIONAL MODULE

ALGORITHM

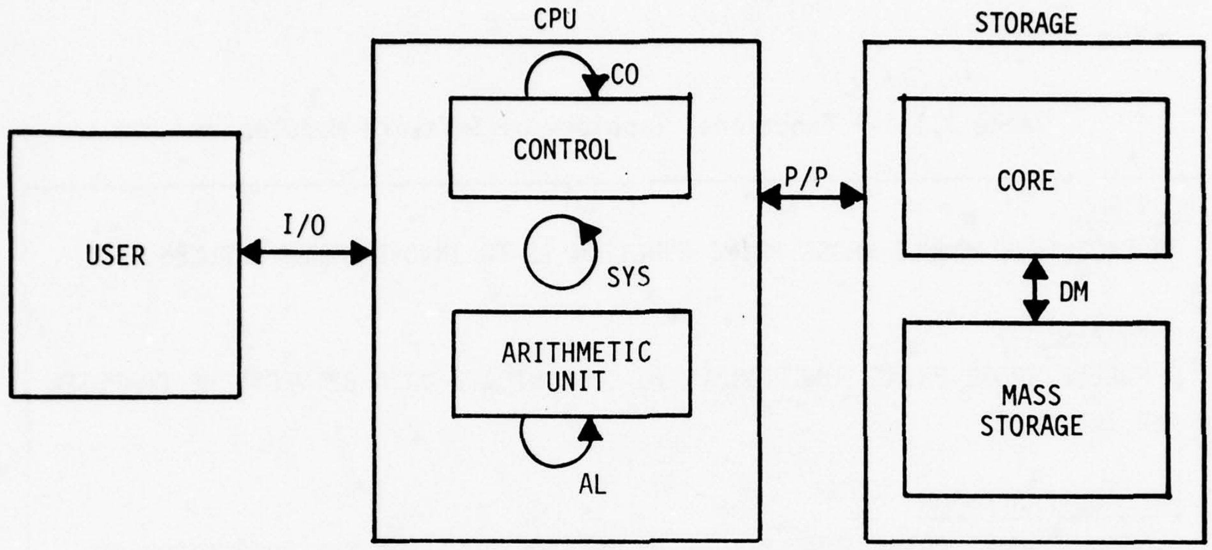
A MODULE WHOSE PRIME FUNCTION IS COMPUTATION

DATA MANAGEMENT

A MODULE WHOSE PRIME FUNCTION IS TO CONTROL THE FLOW OF DATA WITHIN THE COMPUTER

SYSTEM

A MODULE WHOSE FUNCTION IS THE SCHEDULING OF SYSTEM RESOURCES FOR OTHER MODULES



LEGEND	IDENTIFIER
CONTROL	CO
INPUT/OUTPUT	I/O
SYSTEM	SYS
ALGORITHM	AL
PRE/POST PROCESSOR	P/P
DATA MANAGEMENT	DM

Figure 2.1.1-1 Rationale for Functional Categorization

the complexity or difficulty of the module.

Recently a number of structural measures have been proposed as predictors of software reliability [MCCJ77]*. These measures include the complexity of logic flow, depth of interactive nesting, number of "GOTO's", etc. These measures were not applied since they were not available in the software error data base but certainly should be considered in future efforts.

2.1.2 PROBLEM PARAMETERS

In the analysis performed, the number of problems a software module has is used as the measure of software reliability. The more problems the lower the reliability. So the definition of what is a software problem determines what is meant by software reliability. Each of the projects had a formal method for recording software problem reports and these form the basis for the succeeding analysis.

The period of collection varies between the projects. Ideally software problems would be collected during the entire development and in operation. This was not the case but sufficient data was collected to indicate the reliability of various software modules in almost all cases. Had the periods of collection been relatively more consistent, the analyses across projects would have been more significant.

The errors have been classified according to the typology developed in [THAT76]. The classifications are given in Table 2.1.2-1. The typology was used by each of the five contractors to classify their respective software problem reports. Since the typology reflected the type of project from which the typology was developed each of the other contractors had varying success with its use. Their major objections were that there were no standards or criteria for categorization and that the typology was somewhat specific to the command and control system used for the development of the typology.

*See References following page 4-3.

Table 2.1.2-1 TRW Typology of Software Problems

A	COMPUTATIONAL	L	USER REQUESTED CHANGES
B	LOGIC	M	PRESET DATA ERRORS
C	INPUT/OUTPUT	N	GLOBAL VARIABLE DEFINITION
D	DATA HANDLING	P	RECURRENT ERRORS
E	OS/SYSTEM SUPPORT	Q	DOCUMENTATION
F	CONFIGURATION	R	REQUIREMENT COMPLIANCE
G	ROUTINE/ROUTINE INTERFACE	S	UNIDENTIFIED
H	ROUTINE/SYSTEM INTERFACE	T	OPERATOR
I	TAPE PROCESSING	U	QUESTIONS
J	USER INTERFACE	V	HARDWARE
K	DATA BASE INTERFACE		

Typical of the four projects that had to use the typology are the following comments:

> Many of the categories were self-explanatory, while many others were subject to interpretation. The task of interpretation would have been much easier had a description of the categories been documented. Such documentation, possibly a brief one sentence description of each sub-category, would have made the job of the analyst easier.
>
> It would help assure uniform application among different analysts. Categories which seem obvious to the person who developed them on the basis of observed errors are often obscure to the person using them. In fact, it would seem that documentation, although sometimes apparently superfluous, is a necessary part of the task of developing a tool to be used outside the domain of the developers. [FRIM77]*.

Another difficulty with the typology is that there is no differentiation between causative and symptomatic problems. A problem can be classified by either the way a problem exhibited itself or the actual cause of the problem. An example of this type of problem is a program that does not check for the end of tape marks. The problem can either be reported as a tape processing problem (I) or a logic problem (B). Another example is a routine call to another routine passing it an out-of-range parameter. The called routine performs an incorrect calculation as a result. Is the error in the calling string of the first routine, a lack of input checking in the called routine, or a computational error in the called routine? This problem also manifested itself in the categories of user requested changes and recurring problems. Neither of these categories describe the cause of the problem if there is one.

The seriousness of a software problem is a major concern of software maintenance. The goal is to have a few problems and for these problems to be not very serious. The seriousness of a software problem can be viewed in two ways. One is the criticality of the software, how immediate is a repair required, while the other is the difficulty of the repair.

*See References following page 4-3.

The criticality of a software problem can be rated on a four level scale. A critical problem is a problem whose correction is required for the immediate function of the software. A medium serious problem is a problem whose correction is necessary for future function of the software. A problem with low criticality is a problem whose correction is required for functioning of the software as designed, but not for immediate use. An improvement is an enhancement in the function of the software. These problem ratings are shown in Table 2.1.2-2.

The difficulty of a correction to a software problem can be determined by the amount of resources required to correct the problem. The required resources can be measured by the number of manhours required to correct the problem. This is probably the best indicator of the expended resources but requires very careful bookkeeping. The quantity used in this report is the length of time between the formal recording of the problem and the recording of the correction to the problem. While this quantity may not truly reflect the difficulty it is obviously related to the amount of effort devoted to the correction of the problem. However with problems of equal criticality, (i.e., problems given equal priority to fix) there should be a direct relationship between the number of days a problem report is open and the difficulty of corrections.

2.2 THE PROJECT HISTORIES

The histories of the projects which comprise the data bases for this study will prove useful later in this report in understanding some of the problems relating to the development of error baselines. These data bases are part of the software data repository currently being created by RADC. Such a repository, together with a more fully developed software system and error taxonomies, should prove a valuable tool for the study of the software development process and life cycle concepts currently being investigated by the research community.

The succeeding paragraphs provide summaries of the histories of the projects involved in this study. More complete histories may be found in [THAT76]*,

*See References following page 4-3.

Table 2.1.2-2 Criticality of Software Problems

- CRITICAL - CORRECTION NECESSARY FOR IMMEDIATE FUNCTION OF SOFTWARE
- MEDIUM - CORRECTION NECESSARY FOR FUTURE FUNCTION OF SOFTWARE
- LOW - CORRECTION REQUIRED FOR FUNCTIONING OF SOFTWARE AS DESIGNED, BUT NOT FOR IMMEDIATE USE
- IMPROVEMENT - A CHANGE IN THE FUNCTION OF THE SOFTWARE

[FRIM77], [BAKW77], [RYEP77], [WILH77]*. For contractual reasons, full explanations of the operational and functional characteristics of some of the projects are not provided in the literature.

2.2.1 PROJECT 3

This project is a real-time control system for a land-based radar complex. The system entailed both hardware and software developed by the Project 3 contractor. The development methodology was modular, using JOVIAL/J3 as the primary programming language. However, the executive program, as well as some other modules and subroutines, were written in assembly language.

The hardware configuration consists of a dual processor system, both processors being identical. In operation one processor acts as the Central Processing Unit (CPU), and the other as the Input/Output Control Unit (IOCU). Both processors share common access to the 81,920 common memory locations. Each memory location consists of a 24 bit word. No special reconfiguration is needed for either processor to do the work of the other, i.e., the CPU can become the IOCU and the IOCU can become the CPU without any difficulties.

It is interesting to note that this project made use of seven software development tools. These included the following:

(1) Cross Compiler
(2) Compiler Support Software
(3) Cross Assembler
(4) Digital Simulator of the Object Computer
(5) Operating System with Debug Package
(6) Digital System Simulator
(7) Data Collection/Reduction Software

Actual development of the software took place on a dedicated UNIVAC 1108 host system and item (4) above, the Digital Simulator, acted as the test simulator of the project computer.

*See References following page 4-3.

The software system consisted of the Executive, made up of five primary functional units - a Task Manager, Memory Manager, I/O Manager, System Auditing Function and Centralized Error Processor - and 109 application modules. A total of 136,707 lines of code were involved in the development.

Software problem reports were collected during unit testing integration and operational testing in the field. Each of these reports was classified by a programmer who had worked on the project according to the problem typology developed by TRW. This classification was done <u>after</u> the project was completed at the request of RADC. There were 2,165 problem reports collected over a period of 37 months.

The modules which comprised this system were categorized using the functional categories defined in Section 2.1 (as far as possible). Twenty-three modules contained no information about their function and were placed in the undetermined category. These modules accounted for over half the total lines of source code for this project (Table 2.2.1-1).

Each software problem was assigned to a particular module and were included in the subsequent analysis (Table 2.2.1-2).

2.2.2 PROJECT 2

Project 2 consists of an avionics control system comprising five subsystems, a control and displays subsystem, a hardware test monitor, two unspecified system functions (A and B) and an executive function which schedules the other subsystem functions. Two other computers provided system and subsystem simulators during the project to provide a test bed environment. The software was written in JOVIAL/J3B and assembly. There were approximately 80,000 lines of assembly and 40,000 lines of JOVIAL code. The system was composed of 69 modules.

Software problem reports were collected during module verification, intermodule compatibility testing and systems validation. These reports were classified according to the TRW error typology after the project was completed at the request of RADC. There were 2,036 problem reports collected during a period of 28 months (see Table 2.2.2-1).

Table 2.2.1-1 Project 3 Software Modules

NUMBER OF MODULES	109
LINES OF CODE	136,707

FUNCTION	NUMBER OF MODULES	LINES OF CODE	NUMBER OF ERRORS
CO	15	16,580	427
IO	6	2,969	108
PP	22	6,102	208
AL	21	10,045	406
DM	22	24,691	826
UNDETERMINED	23	76,320	189

Table 2.2.1-2 Project 3 Software Problems

NUMBER OF PROBLEM REPORTS 2,165

COLLECTION PERIOD

12/72 - 1/76

PHASES DURING COLLECTION

INTEGRATION, ACCEPTANCE, AND OPERATION

CATEGORY	NUMBER	CATEGORY	NUMBER
A	115	L	764
B	382	M	162
C	21	N	45
D	409	P	39
E	4	Q	15
F	18	R	10
G	16	S	77
H	17	T	15
I	0	U	3
J	10	V	11
K	32		

Insufficient information was available to categorize the modules of this project. Analyses that required knowledge of module function could not be performed on this data set (Table 2.2.2-1).

Not every problem could be assigned to a particular module. Only the 1,443 problems which could be ascribed to particular modules were subjected to detailed analysis (Table 2.2.2-2).

2.2.3 PROJECT 1

This project was a large command and control system. The software was written in JOVIAL/J4. The system was composed of 249 modules, of which 77 were written by an associate contractor. There were 115,346 lines of source statements and 80,993 comment lines.

Software problem reports were collected during development test, validation test, acceptance test, integration test and operational demonstration. The project was used by the Project 1 contractor to develop the problem typology. There were a total of 4,519 problem reports (Table 2.3-1, page 2-25) collected over a nine month period.

Only 145 of the modules could be classified as to function. The 77 modules written by the associate contractor had no information about their function and 27 of the Project 1 modules were classified as utility modules (Table 2.2.3-1). Of the 4,490 software problem reports only 4,087 could be ascribed to particular software modules. The other problems either related to data base changes or nonexistant modules (Table 2.2.3-2).

2.2.4 PROJECT 5

This project was the command and control software for the anti-ballistic missile system. The software was written in CENTRAN. The system was composed of 2,413 modules (Table 2.2.4-1). There were 130,592 lines of source code. The functions which these modules performed included radar surveillance, tracking, target classification, radar management and testing, inter-site communication and command and control display functions. The application required both high reliability and availability, as well as fault-tolerant software.

Table 2.2.2-1 Project 2 Software Modules

NUMBER OF MODULES	69
LINES OF CODE	124,705

NUMBER OF MODULES	LINES OF CODE	NUMBER OF ERRORS
69	124,705	2,036

Table 2.2.2-2 Project 2 Software Problems

NUMBER OF PROBLEM REPORTS 1,443 (2,036)*

COLLECTION PERIOD

5/73 - 8/75

PHASES DURING COLLECTION

DEVELOPMENT AND OPERATION

CATEGORY	NUMBER	CATEGORY	NUMBER
A	105 (109)	L	119 (161)
B	569 (634)	M	53 (67)
C	22 (28)	N	27 (46)
D	244 (272)	P	47 (148)
E	5 (8)	Q	7 (27)
F	10 (12)	R	121 (144)
G	36 (41)	S	23 (30)
H	2 (3)	T	20 (159)
I	3 (5)	U	1 (19)
J	10 (12)	V	3 (32)
K	14 (17)	X	2 (62)

* Numbers in ( ) are total problems including problems that could not be attributed to some software module.

Table 2.2.3-1 Project 1 Software Modules

NUMBER OF MODULES	249
LINES OF CODE	115,346 (196,339)*

FUNCTION	NUMBER OF MODULES	LINES OF CODE	NUMBER OF ERRORS
CO	30	7,203	527
IO	32	18,716	461
PP	18	10,664	365
AL	65	37,262	1,067
UNDETERMINED	104	41,531	1,667

* With Comments

Project 1 Software Problems

Table 2.2.3-2

NUMBER OF PROBLEM REPORTS 4,087 (4,490)*

COLLECTION PERIOD

6/73 - 2/74

PHASES DURING COLLECTION

DEVELOPMENT AND OPERATION

CATEGORY	NUMBER	CATEGORY	NUMBER
A	335 (342)	L	0 (0)
B	914 (960)	M	262 (501)
C	701 (727)	N	37 (55)
D	584 (605)	P	76 (78)
E	1 (1)	Q	177 (187)
F	83 (83)	R	26 (26)
G	244 (248)	S	21 (21)
H	30 (30)	T	117 (134)
I	6 (8)	U	76 (77)
J	377 (385)	V	0 (0)
K	20 (22)		

*Total problems are given (), including problems that could not be attributed to some software module.

Software problem reports were collected during unit testing, process and function testing, and system integration. More than 17,000 problem reports were generated, but only the approximately 6,700 that occurred between 1 March 1974 and 1 March 1975 were in the data base provided by RADC. These reports were classified into the TRW typology using a semi-automated method.

There was no information available about the function of particular modules. Only the subsystem to which a module belonged was available in this data set. Another problem was that these modules did not have unique names so problem reports could not be ascribed to a particular module. This problem was caused by the use of slightly modified software modules at different sites. This problem proscribed the use of this data set in most of the subsequent analysis. Tables 2.2.4-1 and 2.2.4-2 provide the data that was available.

2.2.5 PROJECT 4

This project was the on-board guidance, navigation, and control software used for both the command and lunar module of the Apollo space vehicles. The project was written in assembly except for some interpretive code used for mathematical programming. The system was composed of 22 subsystems but the total number of lines of code can only be estimated as between 83,866 and 610,000 (Table 2.2.5-1). The estimate depends on how much code was reused for each Apollo mission.

This system was developed for the special, single purpose computer used during the Apollo missions for flight guidance and control. The programs were hard-wired into the guidance computer and necessitated core memory conservation techniques which might be considered poor practice in other less weight-conscious environments. The resulting programs were difficult to debug, modify or correct.

Software problem reports were collected during the entire operational period of the Apollo missions. During this time 11,728 problem reports were collected (Table 2.2.5-2). These reports were classified by using a preliminary version of the software problem typology developed in [THAT76]*. The two typologies

*See References following page 4-3.

Table 2.2.4-1 Project 5 Software Modules

NUMBER OF MODULES	2,413
LINES OF CODE	130,592
FUNCTION	UNAVAILABLE

Table 2.2.4-2 Project 5 Software Problems

NUMBER OF PROBLEM REPORTS 5,693

COLLECTION PERIOD

3/74 - 2/75

PHASES DURING COLLECTION

DEVELOPMENT

CATEGORY	NUMBER	CATEGORY	NUMBER
A	170	L	188
B	993	M	310
C	454	N	112
D	347	P	820
E	14	Q	796
F	19	R	32
G	123	S	236
H	38	T	26
I	5	U	102
J	29	V	246
K	176	W	457

Table 2.2.5-1 Project 4 Software Modules

NUMBER OF MODULES	22*
LINES OF CODE	83,866 - 610,000**
FUNCTION	

*ONLY SUBSYSTEM DESCRIPTION AVAILABLE

**RANGE OF ESTIMATES FOR TOTAL LINES OF CODE

Table 2.2.5-2 Project 4 Software Problems

NUMBER OF PROBLEM REPORTS 11,728

COLLECTION PERIOD

2/67 - 2/71

PHASES DURING COLLECTION

DEVLOPMENT AND OPERATION

CATEGORY	NUMBER	CATEGORY	NUMBER
A	541	L	780
B	2,217	M	355
C	287	N	851
D	745	P	280
E	14	Q	727
F	1,122	R	57
G	760	S	66
H	683	T	0
I	0	U	0
J	42	V	2,123
K	79		

are the same as far as major categories are concerned, which were all that were used in this report. The distribution of problems is given in Table 2.2.5-2.

2.3 LIMITATIONS

There were several shortcomings in the software project data base which limited the types of analyses that could be performed. Table 2.3-1 provides a cross project comparison of the data provided.

As already mentioned none of the data bases contained any true structural information about the software modules. The data bases contained at most simple descriptions of the modules.

Only the Project 3 and Project 1 software modules could be categorized by function. In addition only about half the modules in these two cases could be categorized bacause of insufficient information.

In several of the projects the software problem reports either could not be ascribed to a particular module or were ascribed to a nonexistant module. These problem reports were eliminated from most of the subsequent analyses.

On the whole the analysis was more driven by what information was available than what analysis should be done.

Table 2.3-1 Contents of Software Problem Data Base

PROJECT	# OF MODULES	SIZE	LANGUAGE	MODE OF CONSTRUCTION	COMPLEXITY	FUNCTION	STRUCTURAL DATA	# OF SPRs	MODULE AFFECTED	ERROR CODE	DATE OPEN	DATE CLOSED	DAYS OPEN	CRITICALITY	PHASE OF DEVELOPMENT
1	249	X	X	X		X	X	4,519	X	X	X	X	X	X	X
2	69	X	X	X		X		2,036	X	X	X	X			X
3	109	X	X	X	X	X		2,165	X	X	X	X	X	X	X
5	2,413	X	X	X			X	5,693	X	X	X	X			X
4	(22)*	X	X	X				11,728		X		X			X

*SUBSYSTEMS

3.0 ANALYSIS OF THE DATA

The analysis of the data bases provided aims primarily at the prediction of reliability based on empirical data using statistical methods. The approach is phenomenological, relating parameters of software, for example the functional typology given in Table 2.1.1-1, with the observed data.

3.1 ERROR RATES

Predicting the number of problems which may be incurred with a particular software module is an important aspect of reliability theory. This importance is reflected in the life cycle concept, which can be considered temporarily to be divided into two phases, the development phase and the operations and maintenance (O&M) phase.

The software management has two main tasks, control and planning. Within the development phase of a project the prior knowledge of likely error rates allows the manager to schedule test resources in the most efficient manner, and to provide the most thorough testing to the software modules most likely to develop problems. Thus planning and control in development are facilitated. Similarly, during the O&M phase of the life cycle, the allocation of resources to problem areas can be simplified by the likely error rates to be incurred during this period.

The measurement of error rates for this study was by three parameters relating to modules:

- by size of module
- by function
- over time in the life cycle

Overall problem rates are found in Table 3.1.1-1. The results agree well with an error rate of 2 per hundred lines of code given in [NELR78]*, based on a much larger sample.

*See References following page 4-3.

3.1.1 EFFECT OF MODULE SIZE

The size of software modules is commonly thought to be related to a number of software problems. The general feeling is that if a module is twice as long as a similar module it should have on the average at lease twice the number of problems. This hypothesis is not totally supported by the analysis. In the projects shown in Table 3.1.1-1, the correlations in general are low, and do not give us much confidence in stating a casual connection between module size and number of problems, assuming an average module size.

This fact seems to contradict the statement that two problems per hundred lines of code appears to be an empirically valid measure of error rates. And indeed the statement is counterintuitive. If one increases the module size by 100 lines of code, we would expect two more errors to appear. But this ignores the fact that the two error figure is derived on a gross system-level, and that errors can appear between modules, not simply within them.

For this reason an additional analysis was made on the effect of module function by size and problem.

3.1.2 EFFECT OF MODULE TYPE

Modules with different functions might be expected to have different problem rates. The results given in Table 3.1.2-1 show that the error rates for Project 3 do not vary significantly except for the category "undetermined".

Tables 3.1.2-1 and 3.1.2-2 show partial categorizations of modules in Project 1 and Project 3. Note that the aggregate totals indicate error rates in the large as being approximately 1.6 per hundred lines in Project 3 and 3.5 per hundred in Project 1 (Table 3.1.1-1). The module categorization for Project 1 is more complete than that of Project 3. It would seem therefore that the combination of incomplete categorization along with arbitrariness in assigning errors when these occur between modules cause wide variances in the by module type error rates. The aggregated results, based on the project level, smooth over these inadequacies in data and categorization.

Table 3.1.1-1 Problem Rates

PROJECT	MODULES	LINES	SPR'S/100 LINES
3	109	136,707	1.6
2	69	124,705	1.6
1	249	115,346 (196,339)*	3.5 (2.1) *
5	2,413	130,592	1.85
4	22	83,000 - 610,000 **	1.14 - .16 **

* WITH COMMENTS

** RANGE OF ESTIMATES FOR TOTAL LINES OF CODE

Table 3.1.2-1

Project 3 Correlation of Number of Problems with Module Type

Module Function	Slope	Intercept	Correlation
Control	0.0202	6.08	0.732
I/O	0.00989	13.1	0.537
Pre/Post Processing	0.00944	6.83	0.195
Algorithm	0.0114	13.82	0.233
Data Management	0.0058	28.24	0.135
Undetermined	0.00055	6.40	0.228

Table 3.1.2-2
Project 1 Correlation of Number of Problems with Module Type

Module Function	Slope	Intercept	Correlation
Control	0.00726	14.67	0.170
I/O	0.014	1.31	0.757
Pre/Post Processing	0.0225	-0.469	0.723
Algorithm	0.0178	-0.4929	0.777
Data Management	-	-	-
Undetermined	0.0223	-0.0604	0.707

3.1.3 EFFECT OF TIME

The number of problems recorded for each month of the collection periods is given in Figures 3.1.3-1 through 3.1.3-5. As can be seen the number of errors ultimately declines with time but is not a monotonic function. There is an initial increase in the number of problem reports followed by considerable fluctuation during the general decrease in error reports.

These fluctuations may be attributed to two main factors, one which concerns the type of data collected, the other statistical. In general, testing does not begin simultaneously for all software modules. This would account for the initial period during which there is an increase in the number of errors, after which there is a decline in errors. The graphic regularities we see in Figures 3.1.3-1 through 3.1.3-5 tend to support the hypothesis that error data should be classified in time within specific life cycle phase.

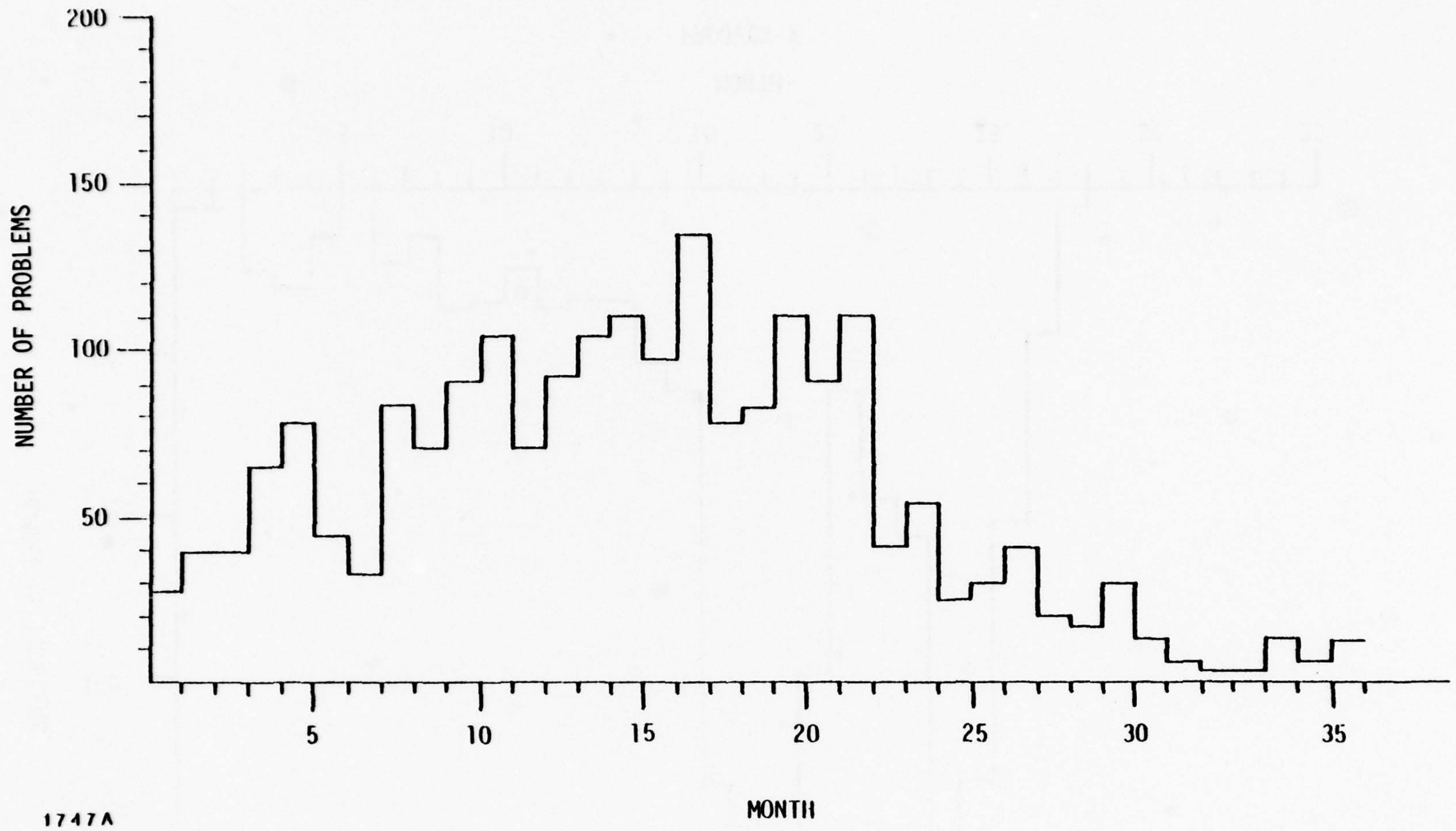
The second point that should be made is that the apparent variances in the graphs are to be expected in any discrete measurement process. It is not possible to continuously find errors.

A further breakdown of the previous graphs is given in Tables 3.1.3-1, -2, -3, -4, -5. Here the type of problem that occurred each month is given. As can be seen there does not seem to be any major differences between the time of occurrence of various types of problems.

3.2 DISTRIBUTION OF PROBLEMS

Although the problem report rates for each of the projects is remarkably similar, there are considerable differences between the projects in the way the problems are distributed in the problem typology (Figure 3.2-1). The most obvious difference is the high peak of type L (user requested changes) problems for Project 3. This reflects the nature of this development as a demonstration project rather than an operational system.

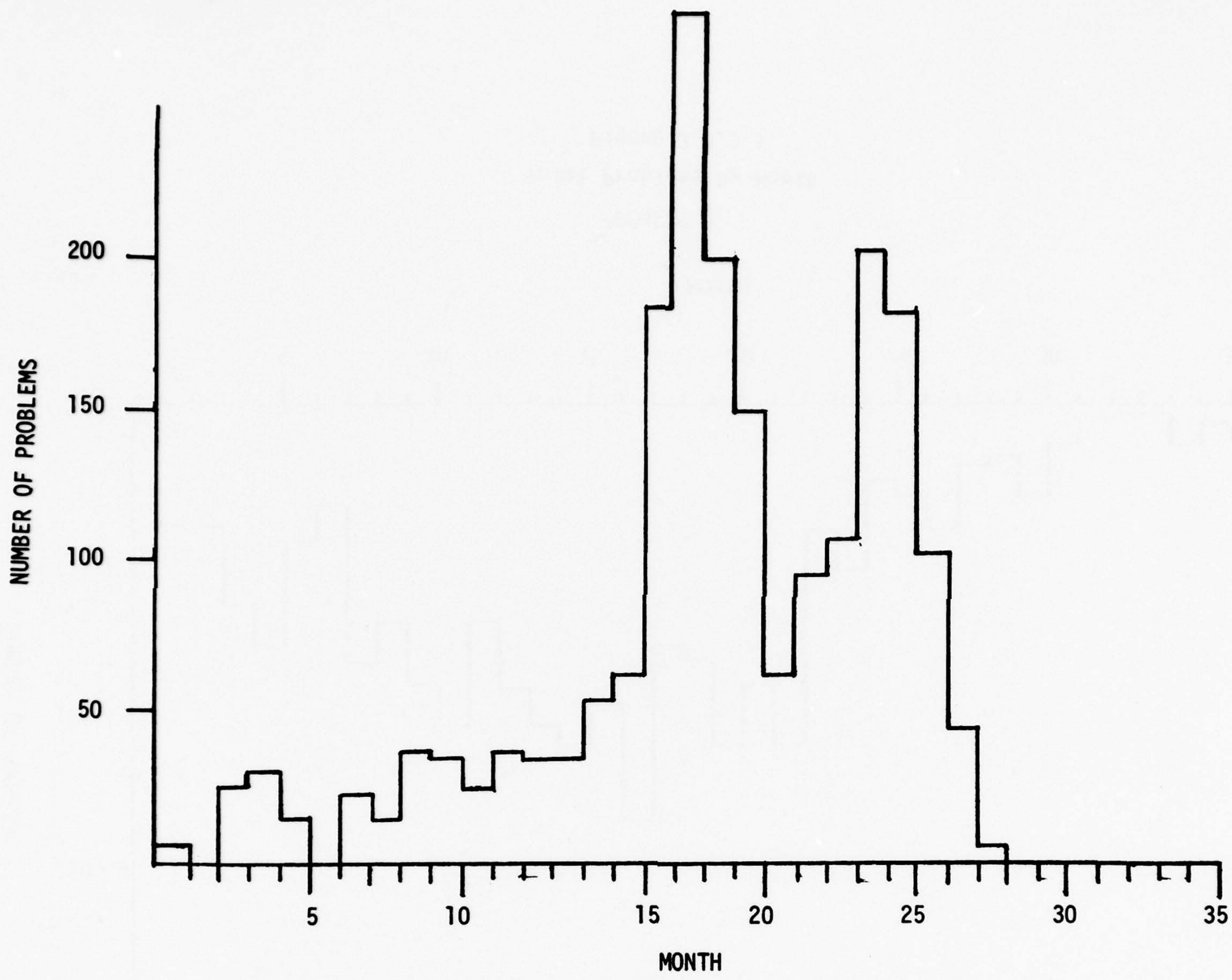
Another major difference is the number of type V (hardware problems) in Project 5 and Project 4. These reflect the special hardware for these projects.



PROJECT 3

Total Problems by Month

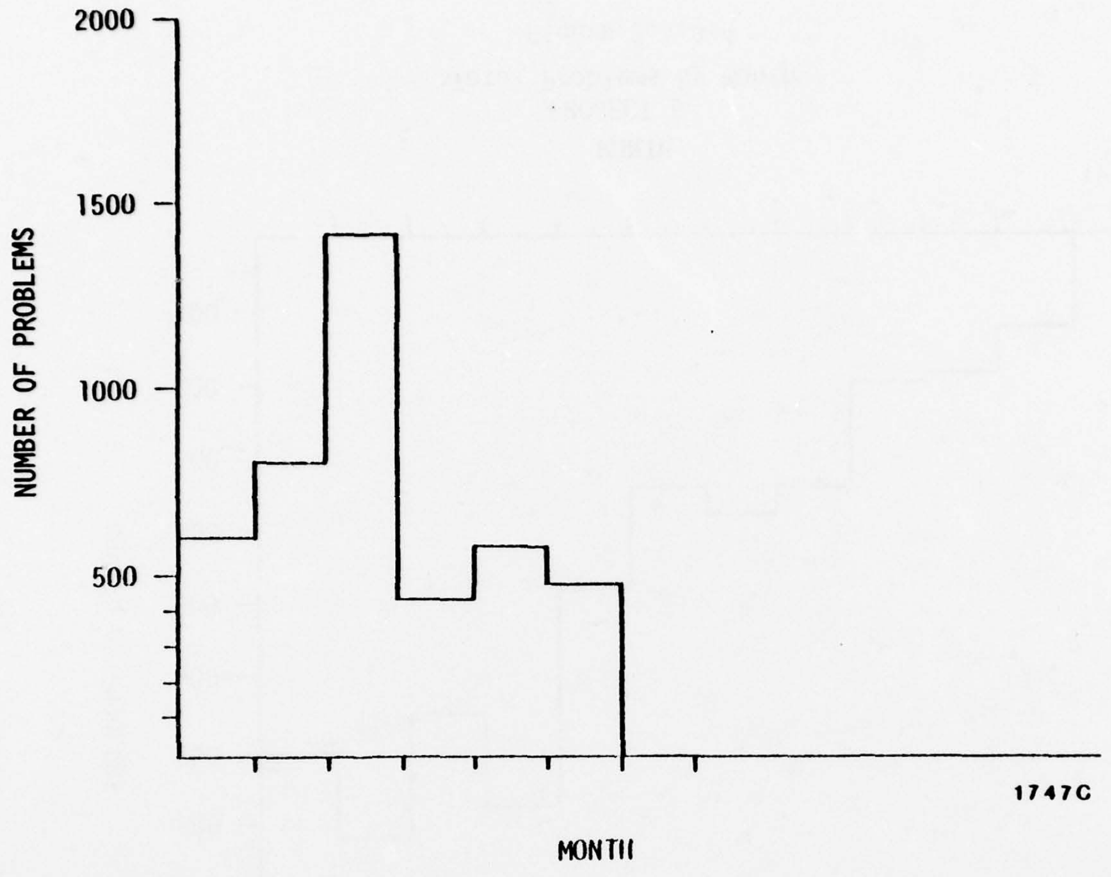
Figure 3.1.3-1



PROJECT 2

Total Problems by Month

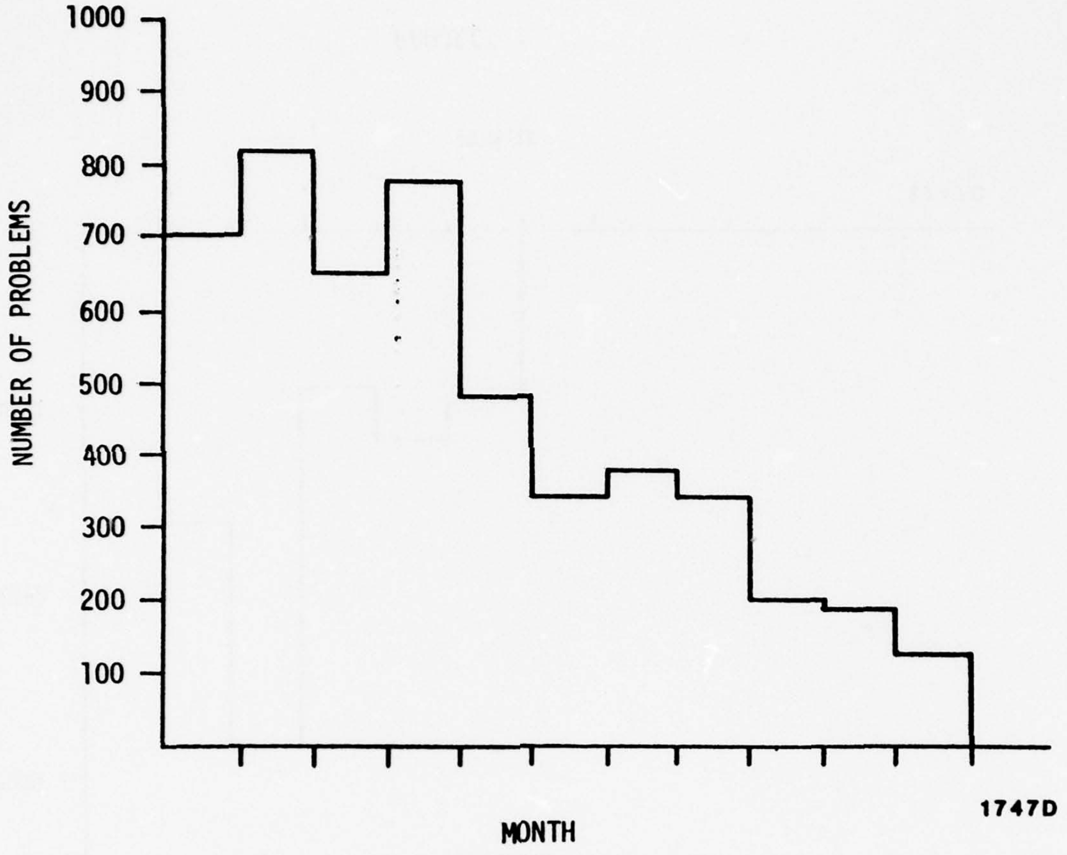
ıgure 3.1.3-2



PROJECT 1

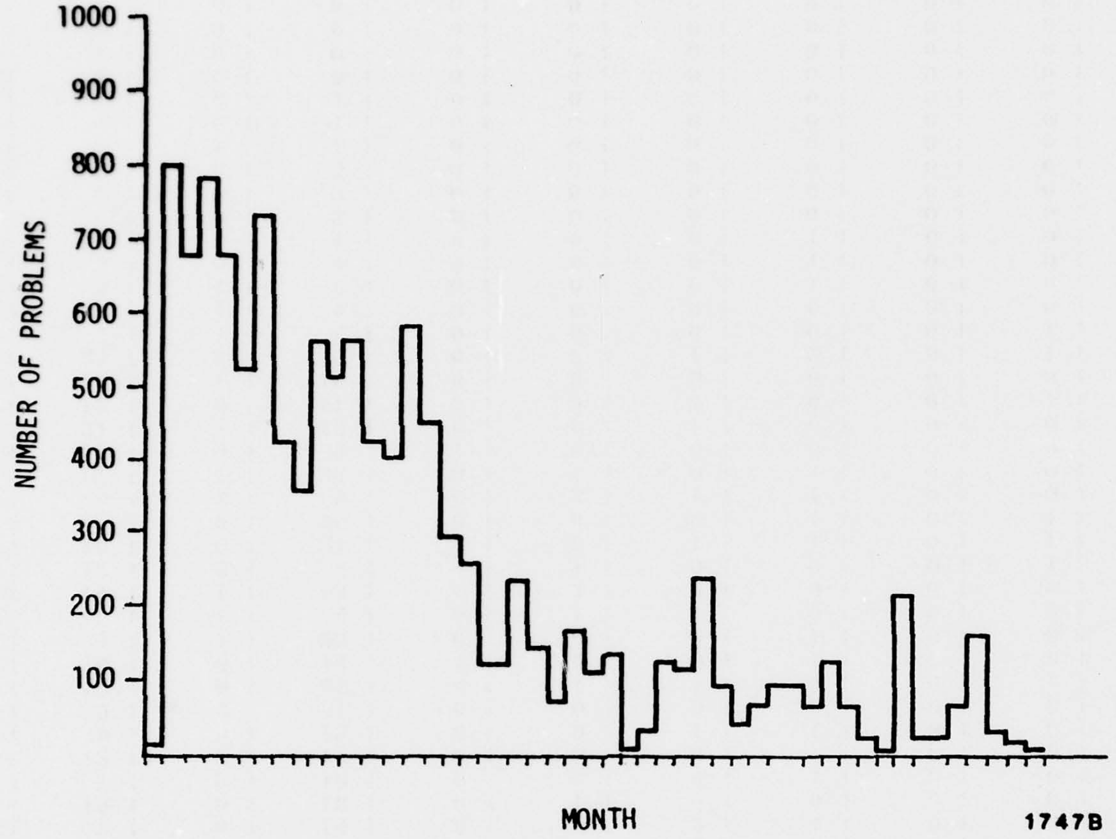
Total Problems by Month

Figure 3.1.3-3



PROJECT 5
Total Problems by Month

Figure 3.1.3-4



PROJECT 4
Total Problems by Month
Figure 3.1.3-5

Table 3.1.3-1 Project 3 Month vs. Problem Type

	A	B	C	D	E	F	G	H	I	J	K	L	M
1972	0	6	0	13	0	0	0	1	0	0	3	3	0
	1	15	0	15	0	0	0	1	0	0	3	2	0
	1	7	0	14	0	0	2	1	0	0	2	10	4
	5	19	0	18	0	1	0	0	0	0	1	16	5
	7	7	2	18	0	4	0	1	0	0	0	22	7
	2	12	2	14	0	0	5	1	0	0	0	13	11
1973	0	9	0	12	0	0	1	1	0	0	0	11	6
	1	9	0	7	0	0	0	0	0	0	3	10	0
	8	17	0	25	0	1	1	0	0	1	4	20	8
	5	10	0	14	1	2	0	1	0	0	2	22	2
	13	21	1	25	0	0	0	1	0	0	0	19	4
	7	17	1	14	0	1	1	0	0	0	1	31	12
—	5	14	1	10	1	1	0	0	0	0	1	27	4
	9	12	2	6	0	1	0	0	0	1	0	54	9
	1	16	0	21	1	2	1	1	0	1	1	34	6
	4	19	0	26	0	0	0	1	0	1	2	54	8
	1	8	2	9	0	2	1	1	0	0	1	37	7
	2	17	1	25	0	1	0	1	0	0	1	58	9
1974	3	14	0	9	1	0	0	0	0	0	0	42	6
	3	21	1	15	0	0	1	0	0	0	0	29	2
	9	18	0	17	0	0	0	0	0	1	0	44	15
	3	18	1	18	0	2	2	0	0	3	0	28	6
	4	23	3	19	0	0	1	2	0	1	3	36	13
—	1	7	1	6	0	0	0	0	0	1	0	20	2
	6	11	1	9	0	0	0	0	0	0	0	26	3
	1	4	1	6	0	0	0	1	0	0	0	8	0
	0	5	0	4	0	0	0	1	0	0	0	10	2
	2	10	0	4	0	0	0	1	0	0	2	16	1
	1	6	0	5	0	0	0	0	0	0	0	15	1
1975	2	1	0	3	0	0	0	0	0	0	0	3	1
	2	5	0	5	0	0	0	0	0	0	0	11	5
	0	0	1	1	0	0	0	0	0	0	0	9	1
	0	0	0	1	0	0	0	0	0	0	0	6	1
	0	1	0	1	0	0	0	0	0	0	0	2	0
	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	1	0	0	0	0	0	0	0	0	0	7	1
1976	1	2	0	0	0	0	0	0	0	0	1	1	0
	1	0	0	0	0	0	0	0	0	0	1	8	0
Total	115	382	21	409	4	18	16	17	0	10	32	764	162

Table 3.1.3-1 Project 3 (Continued)

	N	P	Q	R	S	T	U	V	Total
1972	0	0	0	0	0	1	0	0	27
	0	1	0	0	0	0	0	0	38
	0	0	0	0	0	0	0	0	41
	1	1	0	0	0	0	0	0	67
	1	0	1	0	1	2	0	0	73
1973	0	4	0	1	2	0	0	0	67
	0	1	0	0	0	0	0	0	41
	1	1	0	0	0	0	0	0	32
	2	1	0	0	1	0	0	1	90
	1	2	1	1	2	0	0	1	67
	5	2	0	0	4	0	0	0	95
	2	2	2	0	4	2	0	2	99
	4	0	0	2	4	0	0	0	74
	1	0	1	0	7	0	0	0	103
	3	1	0	1	2	0	0	0	92
	0	1	1	0	2	0	0	1	120
	12	5	1	0	5	2	1	1	96
1974	5	2	0	0	5	2	0	2	131
	3	1	0	2	1	0	0	1	83
	0	2	1	2	4	2	0	1	84
	1	3	1	1	2	2	0	0	114
	0	3	1	0	5	1	1	0	92
	1	2	2	0	6	0	1	0	117
	2	1	2	0	6	0	0	0	49
	0	1	0	0	3	0	0	1	61
	0	1	0	0	4	0	0	0	26
	0	0	0	0	3	1	0	0	26
	0	1	1	0	2	0	0	0	40
	0	0	0	0	1	0	0	0	29
19/5	0	0	0	0	0	0	0	0	10
	0	0	0	0	0	0	0	0	28
	0	0	0	0	1	0	0	0	13
	0	0	0	0	0	0	0	0	8
	0	0	0	0	0	0	0	0	4
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	13
1976	0	0	0	0	0	0	0	0	5
	0	0	0	0	0	0	0	0	10
Total	45	39	15	10	77	15	3	11	2165

Table 3.1.3-2 Project 2 Month vs. Problem Type

		A	B	C	D	E	F	G	H	I	J	K	L	M
	May	0	0	0	0	0	0	0	0	0	1	0	0	0
	Jun	0	0	0	0	0	0	0	0	0	0	0	0	0
	Jul	2	8	0	2	0	0	1	0	1	0	0	1	0
		2	7	0	3	2	0	2	0	0	1	0	0	2
1973		1	4	0	3	0	0	2	0	0	0	0	0	2
		1	0	0	0	0	0	0	0	0	0	0	0	0
		1	6	0	3	0	0	0	0	0	0	0	2	2
		0	5	0	0	0	0	1	0	0	0	0	2	0
		2	7	0	4	0	0	1	0	0	0	0	3	0
		3	13	0	3	0	0	0	0	0	0	1	9	2
		5	9	0	1	0	0	0	0	0	0	0	3	0
		4	11	0	2	0	1	0	0	0	0	0	2	2
		2	15	0	5	0	0	0	0	0	0	0	8	0
1974		0	3	1	1	0	0	1	0	0	1	0	4	2
		2	16	0	8	0	0	0	0	0	0	0	12	3
		7	25	0	4	0	0	2	0	0	0	0	5	3
		6	64	4	26	1	2	0	0	1	1	3	19	7
		20	78	2	49	1	2	5	0	1	1	2	17	7
		11	79	4	32	1	0	2	0	0	0	4	13	4
		5	30	1	19	1	0	5	0	0	0	1	22	3
	Jan	5	19	2	5	0	0	2	1	0	0	0	7	0
		2	29	0	13	1	2	5	0	0	0	2	8	4
		3	33	0	13	1	2	3	2	0	0	1	6	6
		9	64	5	25	0	0	2	0	1	4	1	6	6
1975		10	55	5	28	0	0	5	0	1	1	0	8	4
		4	35	2	13	0	2	1	0	0	1	2	4	6
		2	17	2	8	0	1	1	0	0	0	0	0	2
		0	2	0	2	0	0	0	0	0	1	0	0	0
Total		109	634	28	272	8	12	41	3	5	12	17	161	67

Table 3.1.3-2 Project 2 (Continued)

		N	P	Q	R	S	T	U	V	X	Total
	May	0	0	0	4	0	0	0	0	0	5
	Jun	0	0	0	0	0	0	0	0	0	0
	Jul	0	3	0	1	0	4	0	2	0	25
		0	6	0	2	0	1	0	2	0	30
1973		0	2	0	0	0	1	0	0	0	15
		0	0	0	0	0	0	0	0	0	1
		0	2	0	3	1	2	0	0	0	22
		0	0	1	3	0	1	0	0	1	14
		1	1	0	9	0	8	0	0	0	36
		0	0	0	3	0	1	0	0	0	35
		1	0	0	4	0	0	0	0	1	24
		2	1	0	4	0	4	1	0	0	34
		0	1	0	3	0	1	0	0	0	35
1974		0	1	1	6	0	8	4	1	1	35
		1	1	0	3	0	4	1	1	1	53
		2	2	1	0	0	7	1	2	1	62
		7	7	2	9	3	18	0	2	1	183
		5	18	3	30	2	26	3	2	5	279
		6	10	2	10	2	9	2	1	7	199
		7	17	3	9	3	9	0	1	13	149
	Jan	0	5	5	4	1	6	0	1	0	63
		6	5	0	5	3	5	2	4	0	96
		1	9	2	9	7	7	0	0	2	107
		4	24	2	9	4	14	3	5	13	201
1975		1	24	2	2	1	13	2	5	14	181
		1	7	3	7	2	7	0	3	1	101
		1	2	0	5	1	3	0	0	1	46
		0	0	0	0	0	0	0	0	0	5
Total		46	148	27	144	30	159	19	32	62	2036

Table 3.1.3-3 Project 1 Month vs. Problem Type

	A	B	C	D	E	F	G	H	I	J	K	L	M
	36	130	97	97	0	20	42	0	3	31	5	0	63
	75	169	160	141	0	17	69	15	2	50	13	0	119
	119	253	217	218	0	14	95	10	2	146	4	0	139
1973	47	167	122	40	1	5	14	1	0	70	0	0	68
	41	171	83	52	0	5	17	4	1	67	0	0	52
	23	70	48	55	0	21	11	0	0	21	0	0	59
	1	0	0	2	0	1	0	0	0	0	0	0	0
1974	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	1
TOTAL	342	960	727	605	1	83	248	30	8	385	22	0	501

	N	P	Q	R	S	T	U	V	TOTAL
	14	7	12	9	6	15	15	0	602
	9	11	40	9	1	21	10	0	931
	21	3	57	6	6	70	14	0	1394
	4	18	30	1	0	5	8	0	601
1973	4	16	37	1	5	7	6	0	569
	3	23	11	0	3	16	24	0	388
	0	0	0	0	0	0	0	0	4
1974	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	1
TOTAL	55	78	187	26	21	134	77	0	4490

Table 3.1.3-4 Project 5 Month vs. Problem Type

		A	B	C	D	E	F	G	H	I	J	K	L	M
	MAR	33	171	74	55	2	2	23	10	0	7	36	23	58
	APR	38	116	57	67	1	2	11	7	0	4	37	27	36
	MAY	16	119	72	58	0	3	16	2	0	1	48	14	44
	JUN	13	85	54	20	0	3	8	6	0	1	17	10	29
1964	JUL	11	72	43	24	2	0	11	4	1	1	9	15	18
	AUG	13	95	21	32	0	0	10	0	1	2	8	13	42
	SEP	9	94	26	15	2	4	11	2	0	1	7	10	19
	OCT	8	80	37	21	1	2	9	3	0	2	8	21	24
	NOV	8	69	27	21	3	2	10	1	1	7	1	13	20
	DEC	5	36	20	17	1	1	8	1	1	0	2	12	15
1975	JAN	9	39	14	10	1	0	6	2	0	2	3	16	3
	FEB	7	17	9	7	1	0	0	0	1	1	0	14	2
	TOTAL	170	993	454	347	14	19	123	38	5	29	176	188	310

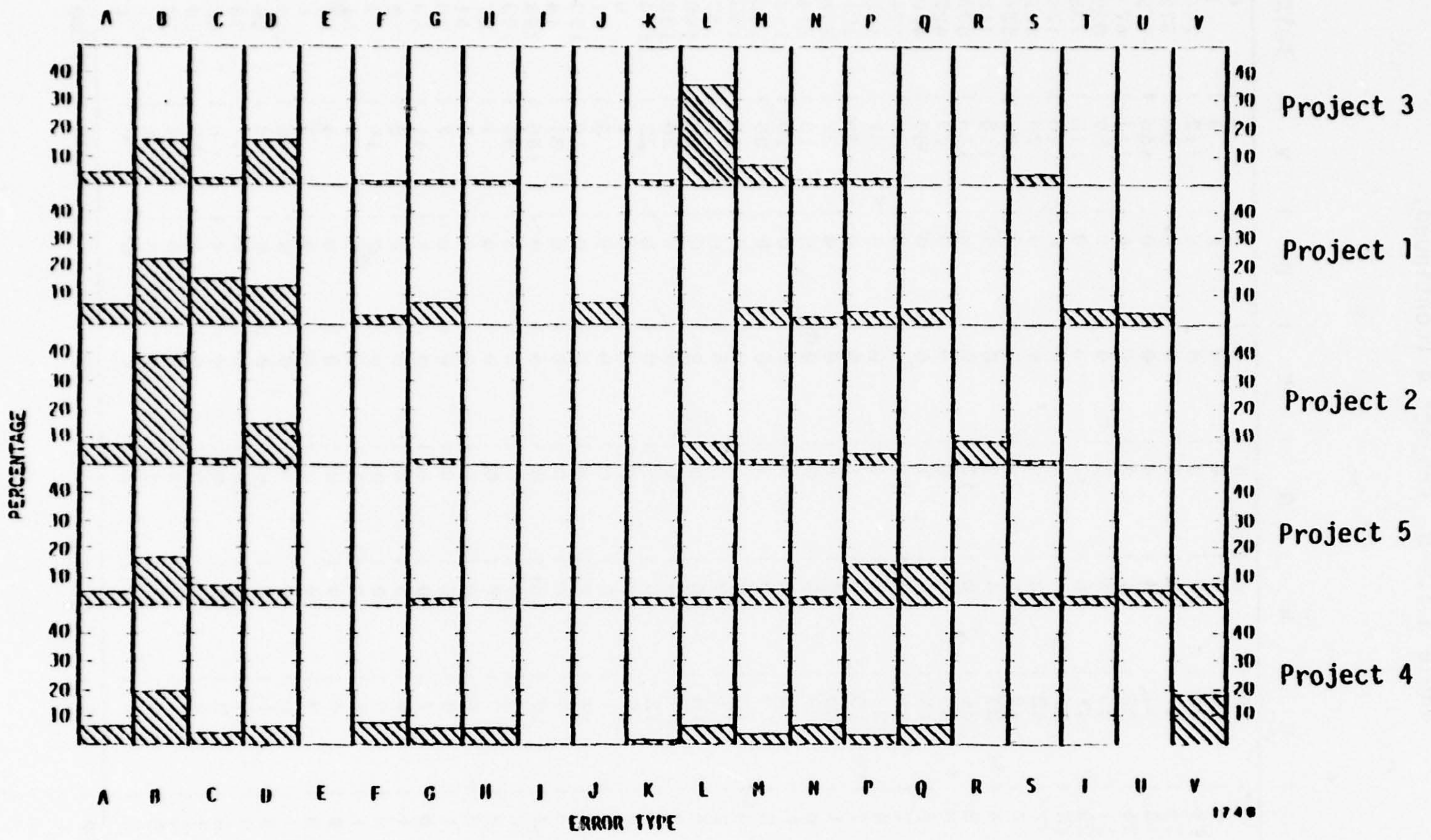
	N	P	Q	R	S	T	U	V	W	X	Y	Z	TOTAL
	23	60	55	5	15	1	8	4	36	0	0	0	701
	15	106	203	8	20	3	6	8	60	0	0	0	832
	12	73	86	3	31	1	13	7	47	0	0	0	666
	7	73	370	2	17	1	4	4	44	0	0	0	768
	19	81	38	2	22	4	54	10	40	0	0	0	481
	6	106	13	4	22	2	7	12	61	0	0	0	470
	10	76	8	0	28	4	0	1	22	0	0	0	349
	8	67	9	3	20	6	2	3	32	0	0	0	366
	6	86	3	0	13	0	5	5	47	0	0	0	348
	2	33	7	0	13	1	2	33	24	0	0	0	234
	1	23	1	4	21	2	1	86	32	0	0	0	276
	3	36	3	1	14	1	0	73	12	0	0	0	202
TOTAL	112	820	796	32	236	26	102	246	457	0	0	0	5693

Table 3.1.3-5 Project 4 Month vs. Problem Type

	A	B	C	D	E	F	G	H	I	J	K	L	M
	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0	0	0	0	0
	13	237	22	63	0	29	6	30	0	2	2	7	1
	44	146	13	58	0	59	30	43	0	0	7	8	13
1967	17	103	37	32	0	112	73	61	0	3	5	44	14
	26	110	18	29	1	85	66	70	0	0	5	33	12
	23	113	21	26	0	47	41	47	0	7	1	38	7
	42	143	26	38	13	93	71	42	0	0	3	83	21
	31	87	8	24	0	43	25	25	0	1	2	15	29
	17	37	8	28	0	52	11	12	0	1	9	27	9
	39	97	16	41	0	64	24	14	0	0	5	48	14
	35	89	14	32	0	43	34	27	0	3	5	41	16
	33	121	14	44	0	32	34	19	0	1	7	16	27
	19	94	6	21	0	29	38	35	0	0	3	25	19
	31	95	13	44	0	16	25	26	0	3	0	26	16
	27	100	10	31	0	49	51	43	0	9	2	43	15
	19	85	4	18	0	37	59	39	0	2	4	28	10
1968	16	54	11	11	0	16	30	22	0	1	3	17	16
	13	40	9	12	0	27	15	13	0	0	4	19	17
	8	26	2	5	0	15	4	7	0	0	1	8	8
	7	26	2	7	0	13	8	21	0	4	0	6	1
	11	44	3	19	0	22	13	20	0	2	7	11	10
	7	25	3	14	0	21	9	4	0	0	0	10	6
	2	12	0	12	0	13	5	1	0	0	0	1	1
	9	31	1	19	0	10	3	3	0	0	0	22	5
	5	20	0	4	0	12	5	1	0	1	0	17	3
	10	17	1	7	0	15	9	7	0	0	2	17	6
	2	1	0	0	0	0	1	0	0	0	0	1	0
	0	10	0	2	0	1	3	0	0	0	0	4	0
1969	5	19	0	12	0	4	9	3	0	0	0	38	2
	5	23	4	15	0	2	9	1	0	1	1	4	2
	2	11	0	6	0	23	7	7	0	0	0	1	3
	3	18	1	6	0	6	0	6	0	0	0	3	1
	0	4	0	2	0	2	2	0	0	0	0	8	1
	1	18	1	1	0	4	1	1	0	0	0	13	4
	1	15	0	2	0	13	2	1	0	0	0	23	6
	4	12	1	2	0	20	2	3	0	0	0	16	4
	2	7	0	6	0	8	2	4	0	0	0	13	2
	2	29	3	10	0	8	5	6	0	0	0	25	3
	0	19	0	7	0	2	9	4	0	0	0	3	3
	0	3	0	2	0	4	2	1	0	0	0	2	5
1970	0	1	0	0	0	1	2	1	0	0	0	0	0
	5	27	3	8	0	30	12	8	0	1	0	46	8
	0	11	3	2	0	1	3	0	0	0	0	1	2
	0	2	0	4	0	1	0	0	0	0	0	0	2
	3	9	2	5	0	11	2	2	0	0	0	7	1
	2	12	7	10	0	24	1	1	0	0	0	6	7
1971	0	10	0	3	0	1	1	3	0	0	1	6	3
	0	3	1	1	0	2	1	0	0	0	0	0	0
	0	0	0	0	0	0	1	0	0	0	0	0	0
Total	541	2216	287	746	14	1122	760	683	0	42	79	788	355

Table 3.1.3-5 Project 4 (Continued)

	N	P	Q	R	S	T	U	V	TOTAL
	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	1
	118	79	28	29	0	0	0	113	779
	86	24	47	8	1	0	0	86	673
	35	12	54	4	0	0	0	166	772
	25	24	52	1	3	0	0	111	671
1967	30	11	28	0	0	0	0	89	529
	19	23	41	6	2	0	0	118	733
	29	15	32	4	6	0	0	48	424
	20	7	42	2	0	0	0	70	352
	42	7	43	1	11	0	0	90	556
	33	4	24	0	3	0	0	115	518
	39	2	36	0	17	0	0	114	556
	27	8	24	0	0	0	0	79	427
	30	6	19	0	1	0	0	50	400
	53	7	36	0	11	0	0	87	574
1968	19	4	24	1	4	0	0	91	448
	25	7	15	0	1	0	0	36	281
	10	1	28	0	1	0	0	27	236
	8	0	9	0	0	0	0	13	114
	3	0	2	1	0	0	0	15	115
	22	3	9	0	3	0	0	39	238
	11	3	13	0	1	0	0	16	143
	6	2	1	0	0	0	0	11	67
	21	6	10	0	0	0	0	20	160
	14	2	1	0	0	0	0	20	105
	6	2	12	0	1	0	0	22	134
	0	0	0	0	0	0	0	1	6
	0	7	1	0	0	0	0	3	31
1969	9	2	4	0	0	0	0	16	123
	10	5	1	0	0	0	0	31	114
	12	2	10	0	0	0	0	156	240
	2	0	6	0	0	0	0	41	93
	8	0	9	0	0	0	0	1	37
	6	0	6	0	0	0	0	5	61
	12	1	5	0	0	0	0	7	88
	7	0	7	0	0	0	0	11	89
	7	1	0	0	0	0	0	8	60
	2	0	8	0	0	0	0	19	120
	2	1	6	0	0	0	0	11	61
	1	0	0	0	0	0	0	4	24
1970	0	0	0	0	0	0	0	1	6
	17	1	8	0	0	0	0	65	239
	1	0	1	0	0	0	0	4	29
	1	0	9	0	0	0	0	11	30
	9	0	3	0	0	0	0	6	60
	13	1	9	0	0	0	0	67	160
	1	0	1	0	0	0	0	9	39
1971	0	0	3	0	0	0	0	0	11
	0	0	0	0	0	0	0	0	1
	851	280	727	57	66	0	0	2123	11728



Problem Distribution by Project

Figure 3.2-1

The other projects generally did not record hardware problems using the same recording methods used for software problems.

This distribution of problems by the type of software module (Figure 3.2-2, 3.2-3) shows great consistency within a project. The difference in problem distribution between control modules and I/O modules within the same project is considerably less than between the two sets of control modules in different projects. The great similarity of problem distributions for different types of modules can be accounted for by either (a) modules of different functional type are more greatly affected by the type of project than by their function or (b) the methods used to record and classify problem reports vary more between the projects than the variance caused by module function.

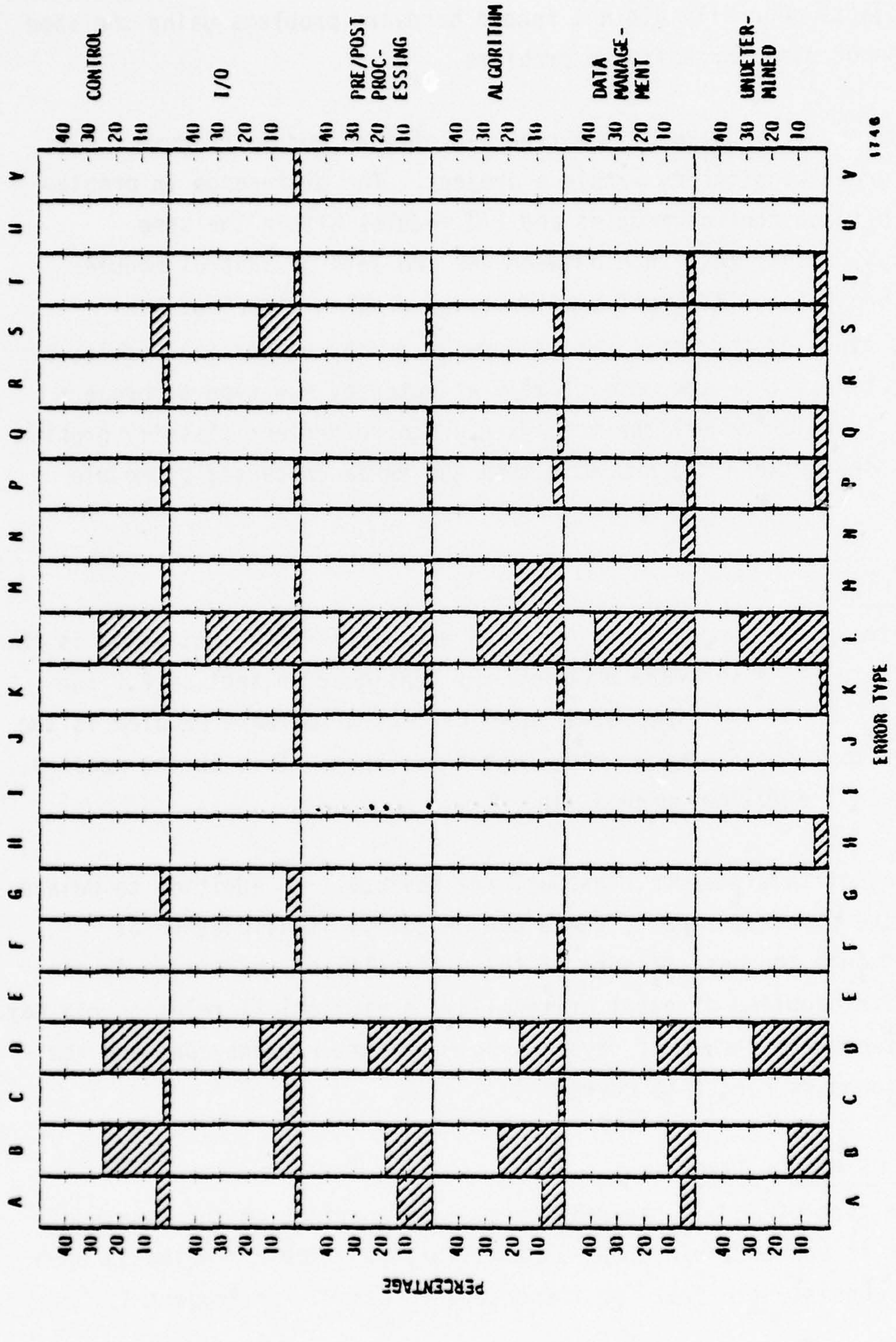
3.3 TIME-TO-FIX

A major software parameter that has not been given sufficient attention is the time necessary to fix a software problem. As mentioned in section 2.3 the only data available on the time required to correct a software problem is the number of days that a software problem report was open. This is the measure that was used in the following analysis.

The limitations of this measure, however, are obvious. In addition to delays in making up the physical report, there can be delays in allocation of resource. Although the best measure of the difficulty of correction is man-hours spent with problems of equal criticality, a statistical relationship may be assumed between the number of days a problem report remains open and the number of personhours needed to correct it.

3.3.1 EFFECT OF MODULE TYPE

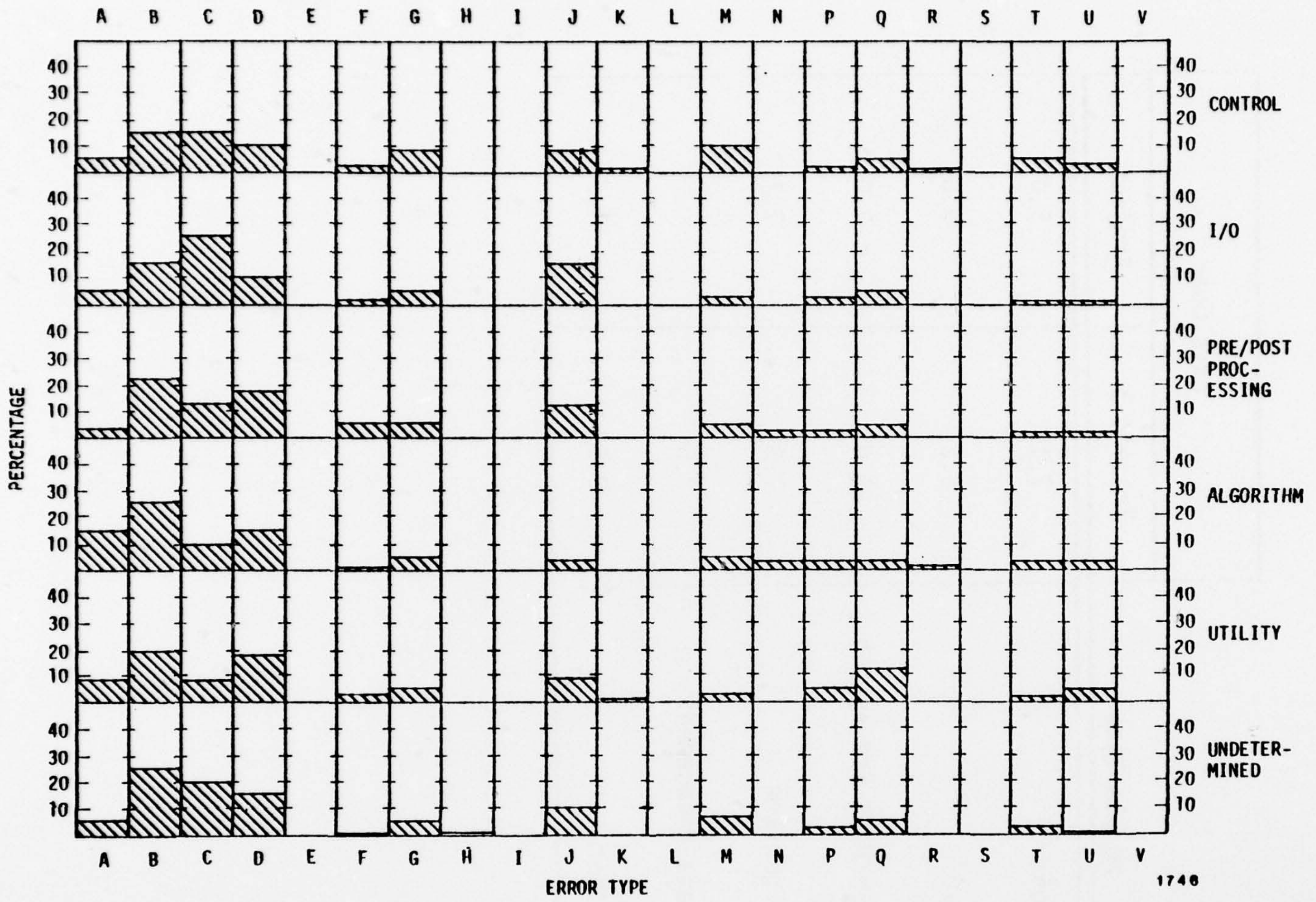
In general the type of module has relatively little effect on the length of time a problem is open. Table 3.3.1-1 shows that the time a problem is open is relatively consistent except for the category "other" for Project 1.



Project 3

Distribution of Problems by Module Type

Figure 3.2-2



Project 1

Distribution of Problems by Module Type

Figure 3.2-3

Table 3.3.1-1 Time-to-Fix by Module Type

Module Type	Days Open	
	Project 3	Project 1
Control	7.1	5.0
I/O	9.8	9.0
Pre/Post Processing	8.0	7.0
Algorithm	7.8	7.2
Data Management	7.6	
Other	9.6	11.7

the average problem remains open from 7 to 9 days.

3.3.2 EFFECT OF ERROR TYPE

A comparison of the time required to resolve software problems as a function of problem type is given in Figure 3.3.2-1. It can be seen that the time required to resolve problems varies considerably with different types of problems but no clear trend between projects is evident.

This may be due to the problems we have previously discussed concerning the adequacy of the error typology and the difficulty associated with the categorization of errors. The lack of trend, the variance, may be due to the non-uniform assignment of errors both across and within projects.

3.4 CROSS PROJECT VALIDITY

As can be seen from the project comparisons in this section, there is considerable variation between projects. The factors causing this variability between these projects cannot be determined from the data available in the software problem data base. The values for problem rates and error distributions derived from these projects can best be used as examples of the range of variability rather than normative values.

The gross rates for the projects are the most consistent values that can be derived from this study. The distribution of the majority of software problems into just a few problem categories is also consistent through all projects.

Tables 3.4-1 through 3.4-3 compile error rate data associated with a project undertaken at GE/Sunnyvale. This was a large command and control system consisting of these subsystems - a command assembly subsystem, a data base management subsystem, and a report generation subsystem. This system has an operational history which we have analyzed. Again there seems to be a consistency associated with gross error rates. This leads us to suspect that such aggregate project-level data are the only meaningful figures which can be

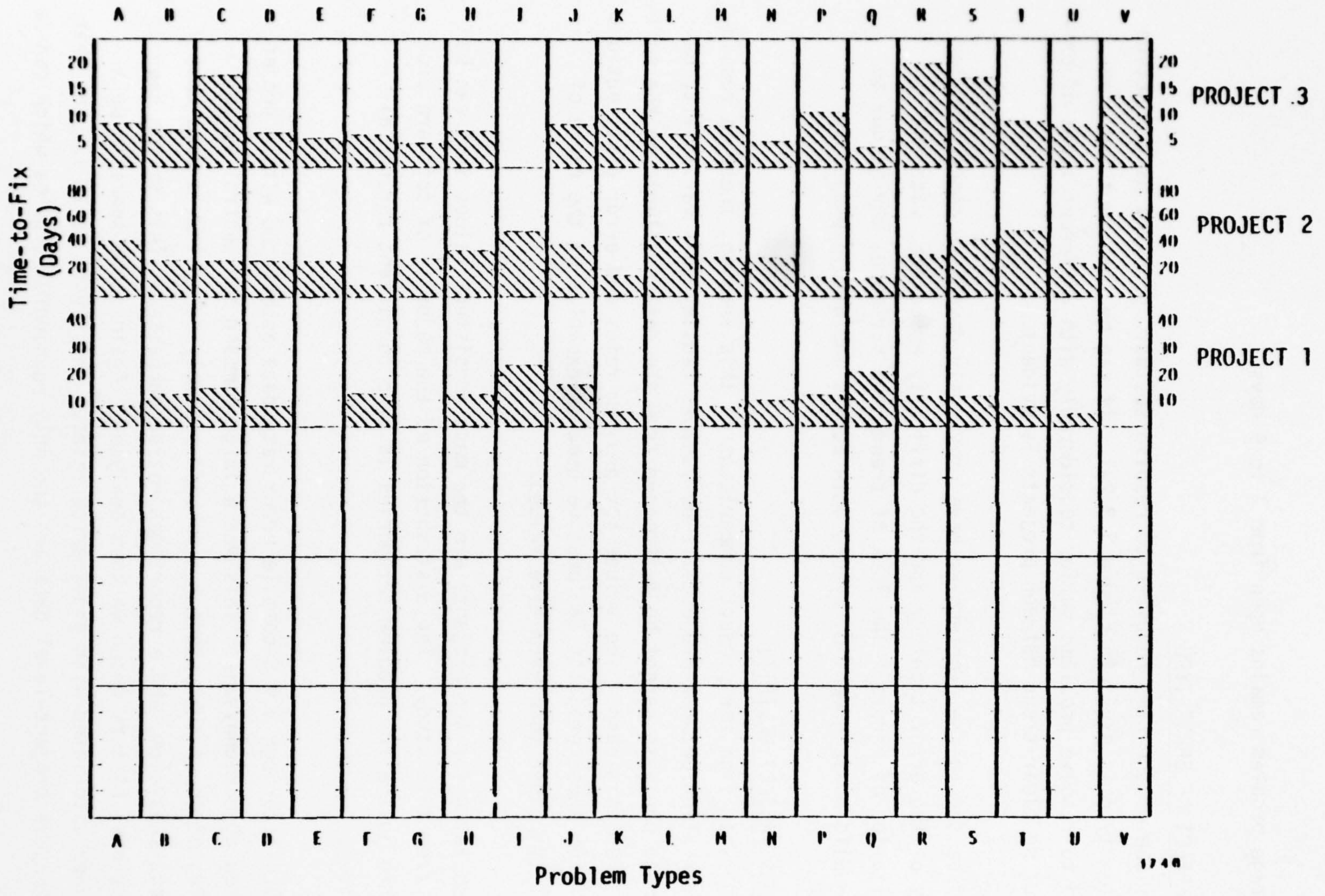


Figure 3.3.2-1 Time-to-Fix by Problem Type

Table 3.4-1

Subsystem 1

TYPE	NO. OF MODULES	NO. OF LINES OF CODE	NO. OF SPR'S	ERROR RATE SPR'S/100 LOC
CONTROL	5	500	7	1.4
DATA MANAGEMENT	13	3840	53	1.4
I/O	10	2060	33	1.6
PRE/POST PROCESSING	8	1270	6	.5
ALGORITHMIC	11	5520	62	1.1
TOTAL	47	13090	160	1.2

Table 3.4-2

Subsystem 2

TYPE	NO. OF MODULES	NO. OF LINES OF CODE	NO. OF SPR'S	ERROR RATE SPR'S/100 LOC
CONTROL	1	490	4	.8
DATA MANAGEMENT	18	7640	97	1.3
I/O	10	3840	41	1.1
PRE/POST PROCESSING	10	3710	32	.9
ALGORITHMIC	2	1300	9	.7
TOTAL	41	16980	183	1.1

Table 3.4-3

Subsystem 3

TYPE	NO. OF MODULES	NO. OF LINES OF CODE	NO. OF SPR'S	ERROR RATE SPR'S/100 LOC
CONTROL	5	2140	33	1.5
DATA MANAGEMENT	22	5300	79	1.5
I/O	14	1200	14	1.2
PRE/POST PROCESSING	8	1900	18	.9
ALGORITHMIC	2	1180	3	.3
SYSTEM	0	0	0	0
TOTAL	51	11720	147	

derived with the current error typology and data collection methodologies. Before true normative values for software problems can be derived, more data must be collected on more factors affecting software development.

4.0 FUTURE CONSIDERATIONS FOR RELIABILITY DATA COLLECTION

There is a great need in software reliability theory for data collected from actual software developments both to confirm existing models and suggest additional models. Just as physical models are confirmed by experimental data, software models must be confirmed by data taken from actual software developments.

Data collected from small experimental projects cannot illustrate the experience of actual large scale software projects.

One of the most difficult aspects of major software projects is communication between the various groups involved in the development. Methods for the coordination of the many diverse activities involved in major software developments are still being investigated. Only from actual software developments can these problems be investigated.

Because of the high cost of data collection it is prohibitively expensive to collect data to test a single hypothesis. Data collection has usually consisted of collecting whatever was thought necessary or possible. As seen from the comments and analyses of the previous sections this has not always been adequate.

In the future, attention should be paid to the type of analyses to be performed. It is not sufficient to record only the most easily obtained information, if this is insufficient to validate an hypothesis. The information not collected is often the most tantalizing. Some of the items that should be collected are given in Table 4.1-1.

Further needs include a better description of how data collection should be performed. Classification of problems is often a difficult task that could be made easier by strong criteria for the classification. An additional need is standard definitions of terminology. Only by using standard terminology can there be consistent interpretation of the results from different projects.

Table 4.1.1

Parameters for Data Collection

(1) System Description
(2) Duration of Each Phase
(3) Management Methods
(4) Design Methods
(5) Coding Methods
(6) Test Methods
(7) Types of Computers Used
(8) Languages Used
(9) General Module Description and Function
(10) Problems for Each Module
  a. Type
  b. Method of Correction
  c. Date of Occurrence
  d. Criticality of Problem
  e. Date of Correction
  f. Difficulty of Correction
  g. Effects of Correction on Other Modules
  h. Manpower Expended on Correction
(11) Structural Measures of Modules
  - Module Length
  - Statement Mix
  - # of Variables
  - Complexity

One last comment on data collection that may now be made with current knowledge. One of the major factors influencing the quality of the data collected is the motivation of the development team to provide the data. A motivating influence is the usefulness of the data to the development team during the development (i.e., real-time feedback). Thus it is important to make the data collection effort beneficial to the developers as well as to the reliability analyst. A vehicle to provide the benefits of the data collection are the preliminary baselines that have been established through this and other studies.

REFERENCES

[BAKW77] Baker, W.F., "Software Data Collection and Analysis: A Real Time System Project History", IBM Corporation, RADC-TR-77-192, June 1977. (A041644)

[FRIM77] Fries, M.J., "Software Error Data Acquisition", Boeing Aerospace Company, RADC-TR-77-130, April 1977. (A039916)

[LLOD77] Lloyd, D., Lipow, M., Reliability: Management, Methods, and Mathematics, Prentice-Hall Inc., Englewood Cliffs, NJ, 1977.

[MCCJ77] McCall, J.A., Richards, P.K., Walters, G.F., "Factors in Software Quality - Concepts and Definitions of Software Quality". General Electric Company, RADC-TR-77-369, Vol I, November 1977. (A049014)

[MOTR77] Motley, R.W., Brooks, W.D., "Statistical Prediction of Programming Errors", IBM Corporation, RADC-TR-77-175, May 1977. (A041106)

[MYEG76] Myers, G., Software Reliability: Principles and Practices, John Wiley & Sons, NY, 1976.

[NELR78] Nelson, R., "Software Data Collection and Analysis (Draft-Partial Report)", RADC, September 1978.

[RYEP77] Rye, P., et al, "Software Systems Development: A CSDL Project History", The Charles Stark Draper Laboratory, Inc. RADC-TR-77-213, June 1977. (A042186)

[THAT76] Thayer, T.A., et al, "Software Reliability Study". TRW Defense and Space Systems Group, RADC-TR-76-238, August 1976. (A030798)

[WILH77] Willman, H.E., Jr., "Software Systems Reliability: A Raytheon Project History", Raytheon Company, RADC-TR-77-188, June 1977. (A040992)

[WOLR74] Wolverton, R.W., "The Cost of Developing Large-Scale Software", IEEE Trans. on Computers, Vol C-23, No. 6, June 1974, pp 615-636.

Appendix A
AID Analysis for Project 1 Structural Data

The Automatic Interaction Dectector Program (AID) is a statistical technique used to identify interaction between several independent variables and a dependent variable. The method is based on successive splitting on the variable which decreases the variance of the dependent variable the most. The method is explained in [SONJ64]*. The result of the analysis is a tree of the binary splittings.

The method was applied to the data in the Project 1 data base. The goal was to achieve a better understanding of the interaction of the various structural parameters given in this report. These parameters are listed in Table A-1.

The parameters given in Table A-1 are not the ideal parameters to use for this method of analysis. Ideally the parameters should not have been previously weighted. For instance the "IF" complexity would be better replaced by a simple count of the number of "IFs".

The results of the analysis is given in Figure A-1. The first division is on executable statements. The modules below 700 executable statements have far fewer problems than those with more than 700 executable statements. The next division of the modules with less than 700 executable statements is on the number of data handling statements. Again there is a major difference between modules with more than 100 data handling statements and those with fewer.

*See Reference following page A-4.

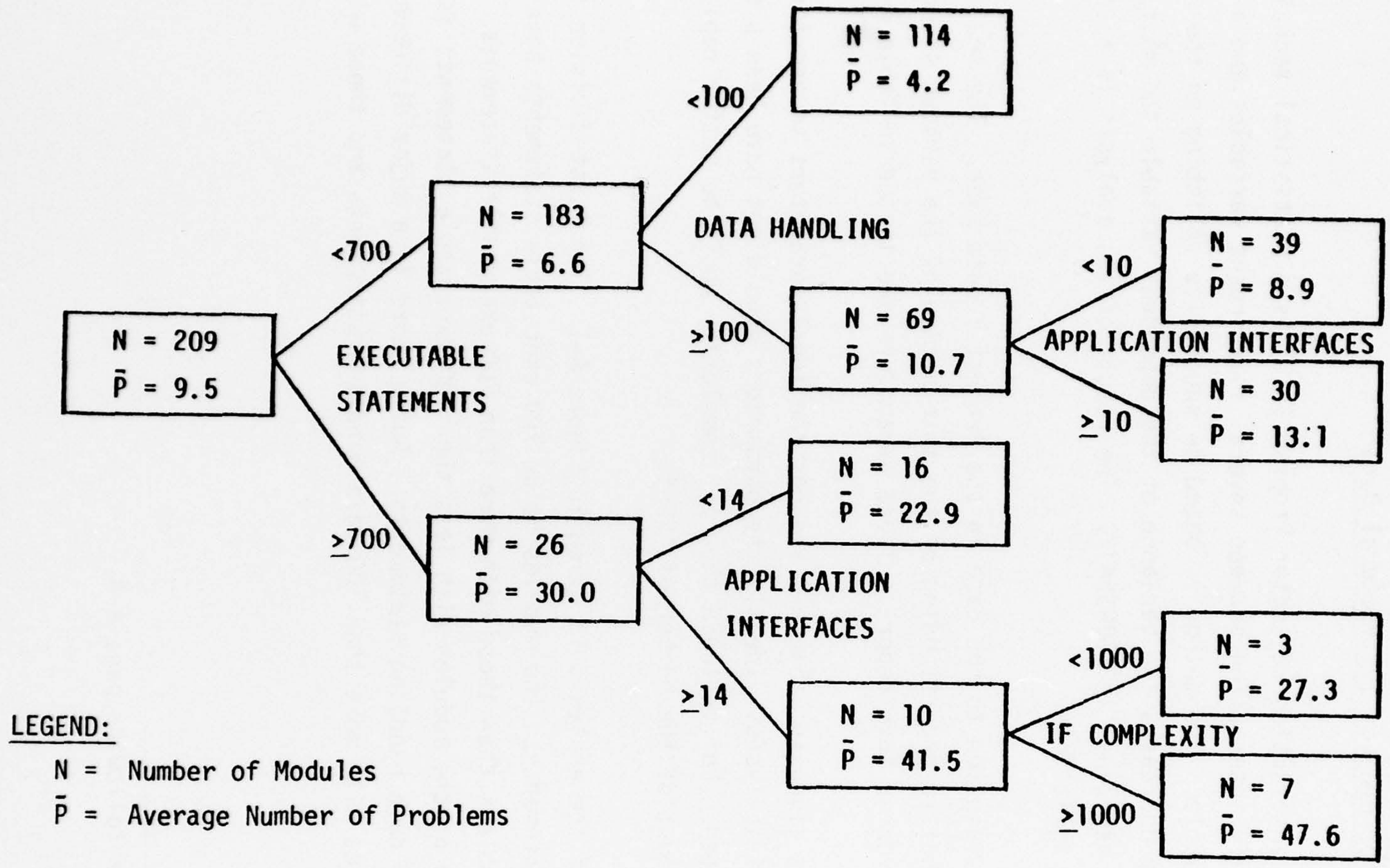


Figure A-1 AID Tree for Software Problems

Table A-1

Aid Parameters

(1) Total Routine Statement

(2) Loop Complexity, which is defined as:

$\sum m_i \ w_i$ where

$$w_i = 4^{i-1} \frac{3}{4^Q - 1} \quad \text{so that}$$

$$\sum_{i=1}^{Q} w_i = 1 \quad \text{and}$$

m_i = number of loops in routine at indentation or nesting level i

w_i = weighting factor

Q = maximum level of indentation in the system

4 = shaping value

(3) IF complexity, which is defined as:

$\sum n_i \ w_i$

where

n_i = number of IFs in routine at indentation or nesting level i

w_i = weighting factor (the same as for loop complexity)

(4) Total Routine Branches

(5) Logical Statements (IF, ORIF, IFEITH)

Table A-1

Aid Parameters (Continued)

(6)	Direct routine interfaces with other applications routines (not a count of calls to other routines).
(7)	Direct routine interfaces with operating system or system support routines (not a count of calls to system routines.
(8)	Routine input/output statements
(9)	Routine computational statements
(10)	Routine data handling statements
(11)	Routine nonexecutable statements
(12)	Routine executable statements
(13)	Total interfaces with other routines
(14)	Total routine comments

APPENDIX REFERENCE

[SONJ64] Sonquist, J.C., Morgan, J.N., the Detection of Interaction Effects, Monograph No. 35, Survey Research Center Institute for Social Research, The University of Michigan, 1964.